This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

A complete list of trademark names appears in a separate “Trademark Information” document.

This is an unpublished work of Mentor Graphics Corporation.
# TABLE OF CONTENTS

About This Manual ............................................................................................. xiii

Related Publications ........................................................................................... xiv
Command Line Syntax Conventions ................................................................... xvi
Acronyms Used in This Manual ......................................................................... xvii

Chapter 1
Introduction ......................................................................................................... 1-1

Features ............................................................................................................... 1-1
Inputs and Outputs .............................................................................................. 1-2

Chapter 2
Command Dictionary .......................................................................................... 2-1

Command Summary ........................................................................................... 2-1
Command Descriptions ....................................................................................... 2-15
Add Atpg Constraints ....................................................................................... 2-16
Add Buffer Insertion ....................................................................................... 2-18
Add Cell Library ............................................................................................... 2-21
Add Cell Models .............................................................................................. 2-22
Add Clock Groups ............................................................................................. 2-26
Add Clocks ....................................................................................................... 2-28
Add Display Instances ..................................................................................... 2-30
Add Display Loop ............................................................................................. 2-34
Add Display Path ............................................................................................. 2-36
Add Display Scanpath ....................................................................................... 2-38
Add Faults ......................................................................................................... 2-40
Add LFSR Connections .................................................................................... 2-42
Add LFSR Taps .................................................................................................. 2-44
Add LFSRs ........................................................................................................ 2-46
Add Mapping Definition ................................................................................... 2-48
Add Mos Direction ............................................................................................ 2-51
Add Mtpi Controller ........................................................................................ 2-53
Add Mtpi Output ............................................................................................... 2-55
Add Net Property ............................................................................................. 2-57
# TABLE OF CONTENTS [continued]

Add Nofaults .................................................................................................. 2-58  
Add Nonscan Instances .................................................................................. 2-60  
Add Nonscan Models ..................................................................................... 2-62  
Add Notest Points ........................................................................................ 2-64  
Add Output Masks ......................................................................................... 2-66  
Add Pin Constraints ....................................................................................... 2-68  
Add Pin Equivalences .................................................................................... 2-70  
Add Primary Inputs ........................................................................................ 2-72  
Add Primary Outputs ..................................................................................... 2-74  
Add Read Controls ......................................................................................... 2-75  
Add Scan Chains ............................................................................................ 2-76  
Add Scan Groups ........................................................................................... 2-78  
Add Scan Instances ........................................................................................ 2-80  
Add Scan Models ............................................................................................ 2-83  
Add Scan Pins ................................................................................................ 2-85  
Add Seq_transparent Constraints .................................................................. 2-88  
Add Sub Chains .............................................................................................. 2-90  
Add Test Points .............................................................................................. 2-94  
Add Tied Signals ............................................................................................ 2-100  
Add Write Controls ........................................................................................ 2-102  
Analyze Control Signals .............................................................................. 2-103  
Analyze Drc Violation .................................................................................... 2-105  
Analyze Input Control .................................................................................... 2-111  
Analyze Output Observe ................................................................................ 2-113  
Analyze Testability ....................................................................................... 2-115  
Close Schematic Viewer .............................................................................. 2-118  
Delete Atpg Constraints ............................................................................... 2-119  
Delete Buffer Insertion ................................................................................. 2-121  
Delete Cell Models ....................................................................................... 2-123  
Delete Clock Groups .................................................................................... 2-125  
Delete Clocks ............................................................................................... 2-127  
Delete Display Instances ............................................................................ 2-128  
Delete Faults ................................................................................................. 2-130  
Delete LFSR Connections ........................................................................... 2-132  
Delete LFSR Taps ......................................................................................... 2-134
## TABLE OF CONTENTS [continued]

- Delete LFSRs ................................................................. 2-136
- Delete Mapping Definition ........................................... 2-138
- Delete Mos Direction ..................................................... 2-142
- Delete Mtapi Controller ................................................. 2-143
- Delete Mtapi Output ..................................................... 2-144
- Delete Net Property ....................................................... 2-146
- Delete Nofaults .............................................................. 2-147
- Delete Nonscan Instances ............................................. 2-149
- Delete Nonscan Models ................................................ 2-151
- Delete Notest Points ..................................................... 2-153
- Delete Output Masks ..................................................... 2-155
- Delete Pin Constraints .................................................. 2-157
- Delete Pin Equivalences ............................................... 2-159
- Delete Primary Inputs ................................................... 2-160
- Delete Primary Outputs ............................................... 2-162
- Delete Read Controls ................................................... 2-164
- Delete Scan Chains ....................................................... 2-165
- Delete Scan Groups ...................................................... 2-166
- Delete Scan Instances .................................................. 2-167
- Delete Scan Models ...................................................... 2-169
- Delete Scan Pins ........................................................... 2-171
- Delete Seq_transparent Constraints ................................. 2-172
- Delete Sub Chains ......................................................... 2-174
- Delete Test Points ........................................................ 2-176
- Delete Tied Signals ........................................................ 2-178
- Delete Write Controls ................................................... 2-180
- Dofile ........................................................................... 2-181
- Exit .............................................................................. 2-183
- Extract Subckts ............................................................. 2-185
- Flatten Subckt .............................................................. 2-186
- Help ............................................................................ 2-187
- Insert Scan Chains ....................................................... 2-188
- Insert Test Logic ........................................................... 2-193
- Mark ............................................................................ 2-200
- Open Schematic Viewer ................................................. 2-202
TABLE OF CONTENTS

Read Subckts Library ................................................................................... 2-203
Redo Display ................................................................................................ 2-204
Report Atpg Constraints ............................................................................... 2-206
Report Buffer Insertion ............................................................................... 2-207
Report Cell Models ...................................................................................... 2-208
Report Clock Groups ................................................................................... 2-210
Report Clocks ............................................................................................... 2-211
Report Control Signals ................................................................................. 2-212
Report Dft Check ......................................................................................... 2-214
Report Display .............................................................................................. 2-218
Report Display Instances ........................................................................... 2-218
Report Drc Rules .......................................................................................... 2-221
Report Environment ..................................................................................... 2-227
Report Faults ................................................................................................ 2-229
Report Feedback Paths ................................................................................ 2-233
Report Flatten Rules .................................................................................... 2-235
Report Gates ................................................................................................. 2-239
Report Lfsr Connections ............................................................................. 2-245
Report Lfsrs .................................................................................................. 2-246
Report Loops ................................................................................................ 2-247
Report Mapping Definition ......................................................................... 2-248
Report Mos Direction ................................................................................... 2-251
Report Mtpi Controller ............................................................................... 2-252
Report Net Properties ................................................................................... 2-254
Report Nofaults ............................................................................................ 2-255
Report Nonscan Instances .......................................................................... 2-257
Report Nonscan Models ............................................................................... 2-259
Report Notest Points .................................................................................... 2-261
Report Output Masks ................................................................................... 2-262
Report Pin Constraints ............................................................................... 2-263
Report Pin Equivalences .............................................................................. 2-265
Report Primary Inputs ............................................................................... 2-266
Report Primary Outputs ............................................................................. 2-267
Report Read Controls ................................................................................ 2-268
Report Scan Cells ......................................................................................... 2-269
Report Scan Chains ...................................................................................... 2-271
# TABLE OF CONTENTS [continued]

- Report Scan Groups ................................................................. 2-272
- Report Scan Identification ....................................................... 2-273
- Report Scan Instances ............................................................. 2-275
- Report Scan Models ............................................................... 2-276
- Report Scan Pins .................................................................. 2-277
- Report Seq_transparent Constraints ........................................ 2-278
- Report Statistics ................................................................... 2-279
- Report Sub Chains ............................................................... 2-281
- Report Test Logic ................................................................. 2-282
- Report Test Points ................................................................. 2-284
- Report Testability Analysis .................................................... 2-286
- Report Tied Signals ............................................................... 2-289
- Report Write Controls .......................................................... 2-290
- Reset State ............................................................................ 2-291
- Ripup Scan Chains ............................................................... 2-292
- Run ......................................................................................... 2-294
- Save Patterns ......................................................................... 2-296
- Save Schematic ..................................................................... 2-299
- Select Object .......................................................................... 2-300
- Set Bist Initialization .............................................................. 2-302
- Set Capture Clock ................................................................. 2-303
- Set Contention Check ............................................................ 2-305
- Set Control Threshold ............................................................ 2-307
- Set Dofile Abort ..................................................................... 2-308
- Set Drc Handling ................................................................. 2-309
- Set Fault Sampling ............................................................... 2-317
- Set Flatten Handling ............................................................. 2-318
- Set Gate Level ....................................................................... 2-322
- Set Gate Report ................................................................. 2-324
- Set Identification Model ........................................................... 2-327
- Set Instancename Visibility .................................................... 2-330
- Set Internal Fault ................................................................. 2-332
- Set Io Insertion ...................................................................... 2-333
- Set Latch Handling .............................................................. 2-336
- Set Lockup Latch ................................................................. 2-338
# TABLE OF CONTENTS [continued]

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Logfile Handling</td>
<td>2-341</td>
</tr>
<tr>
<td>Set Loop Duplication</td>
<td>2-343</td>
</tr>
<tr>
<td>Set Multiple Scan enables</td>
<td>2-345</td>
</tr>
<tr>
<td>Set Net Resolution</td>
<td>2-347</td>
</tr>
<tr>
<td>Set Nonscan Handling</td>
<td>2-348</td>
</tr>
<tr>
<td>Set Observe Threshold</td>
<td>2-349</td>
</tr>
<tr>
<td>Set Pattern Save</td>
<td>2-350</td>
</tr>
<tr>
<td>Set Random Patterns</td>
<td>2-351</td>
</tr>
<tr>
<td>Set Scan Type</td>
<td>2-352</td>
</tr>
<tr>
<td>Set Schematic Display</td>
<td>2-353</td>
</tr>
<tr>
<td>Set Screen Display</td>
<td>2-356</td>
</tr>
<tr>
<td>Set Sensitization Checking</td>
<td>2-357</td>
</tr>
<tr>
<td>Set Stability Check</td>
<td>2-358</td>
</tr>
<tr>
<td>Set System Mode</td>
<td>2-359</td>
</tr>
<tr>
<td>Set Test Logic</td>
<td>2-361</td>
</tr>
<tr>
<td>Set Trace Report</td>
<td>2-364</td>
</tr>
<tr>
<td>Set Zoom Factor</td>
<td>2-365</td>
</tr>
<tr>
<td>Setup LFSRs</td>
<td>2-366</td>
</tr>
<tr>
<td>Setup Output Masks</td>
<td>2-368</td>
</tr>
<tr>
<td>Setup Pin Constraints</td>
<td>2-370</td>
</tr>
<tr>
<td>Setup Scan Identification</td>
<td>2-372</td>
</tr>
<tr>
<td>Setup Scan Insertion</td>
<td>2-380</td>
</tr>
<tr>
<td>Setup Scan Pins</td>
<td>2-385</td>
</tr>
<tr>
<td>Setup Test_point Identification</td>
<td>2-387</td>
</tr>
<tr>
<td>Setup Test_point Insertion</td>
<td>2-395</td>
</tr>
<tr>
<td>Setup Tied Signals</td>
<td>2-399</td>
</tr>
<tr>
<td>System</td>
<td>2-401</td>
</tr>
<tr>
<td>Undo Display</td>
<td>2-402</td>
</tr>
<tr>
<td>Unmark</td>
<td>2-404</td>
</tr>
<tr>
<td>Unselect Object</td>
<td>2-406</td>
</tr>
<tr>
<td>View</td>
<td>2-408</td>
</tr>
<tr>
<td>View Area</td>
<td>2-410</td>
</tr>
<tr>
<td>Write Atpg Setup</td>
<td>2-412</td>
</tr>
<tr>
<td>Write Bist Setup</td>
<td>2-414</td>
</tr>
<tr>
<td>Write Loops</td>
<td>2-416</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS [continued]

Write Netlist ........................................................................................................ 2-417
Write Primary Inputs .......................................................................................... 2-419
Write Primary Outputs ...................................................................................... 2-421
Write Scan Identification ............................................................................... 2-423
Write Subchain Setup ...................................................................................... 2-425
Zoom In ........................................................................................................... 2-427
Zoom Out ......................................................................................................... 2-428

### Chapter 3

#### Shell Commands

Shell Command Descriptions ........................................................................... 3-1
dftadvisor ........................................................................................................... 3-2

### Index
LIST OF FIGURES

Figure 1. DFT Documentation Roadmap ........................................................... xiv
Figure 2-1. MISR pin placement ....................................................................... 2-42
Figure 2-2. DFT Library Model GDFF ............................................................ 2-89
Figure 2-3. Control Example .......................................................................... 2-96
Figure 2-4. Observe Example .......................................................................... 2-97
Figure 2-5. Zoom In and Zoom Out Icons ..................................................... 2-365
Figure 2-6. Control Point Example ................................................................. 2-396
Figure 2-7. Observe Point Example ................................................................. 2-396
# LIST OF TABLES

Table 2-1. Command Summary ................................................................. 2-1  
Table 2-2. Fault Class Codes and Names .................................................. 2-230  
Table 2-3. Report Gate Types ................................................................. 2-242  
Table 2-4. Scan Insertion Invocation Default Pin Names .......................... 2-380
LIST OF TABLES [continued]
This manual contains reference information on the Mentor Graphics DFTAdvisor product. DFTAdvisor is a highly accurate test synthesis tool, which is capable of identifying and replacing sequential elements with their corresponding scan cells, and stitching them together into scan chains. This manual is divided into the following three sections:

- Chapter 1, “Introduction” - briefly describes the features, inputs, and outputs of DFTAdvisor.
- Chapter 2, “Command Dictionary” - contains a command summary table and a detailed description of each DFTAdvisor command.
- Chapter 3, “Shell Commands” - describes the DFTAdvisor invocation command in detail.

The DFT applications use Adobe Acrobat Exchange as their online documentation and help viewer. Online help requires installing the Mentor Graphics-supplied Acrobat Exchange program with Mentor Graphics-specific plugins and also requires setting an environment variable. For more information, refer to the section, “Setting Up Online Manuals and Help” in Using Mentor Graphics Documentation with Acrobat Exchange.
Related Publications

This section gives references to both Mentor Graphics product documentation. Figure 1 shows the Mentor Graphics DFT manuals and their relationship to each other and is followed by a list of descriptions for these documents.

Figure 1. DFT Documentation Roadmap

**Boundary Scan Process Guide** — provides process, concept, and procedure information for the boundary scan product, BSDArchitect. It also includes information on how to integrate boundary scan with the other DFT technologies.

**BSDArchitect Reference Manual** — provides reference information for BSDArchitect, the boundary scan product.

Design-for-Test Common Resources Manual — contains information common to many of the DFT tools: design rule checks (DRC), DFTInsight (the schematic viewer), library creation, VHDL support, Verilog support, Spice support, and test procedure file format.

Design-for-Test Release Notes — provides release information that reflects changes to the DFT products for the software version release.

DFTAdvisor Reference Manual — provides reference information for DFTAdvisor (internal scan insertion) and DFTInsight (schematic viewer) products.

FastScan and FlexTest Reference Manual — provides reference information for FastScan (full-scan ATPG), FlexTest (non- to partial-scan ATPG), and DFTInsight (schematic viewer) products.


Scan and ATPG Process Guide — provides process, concept, and procedure information for using DFTAdvisor, FastScan, and FlexTest in the context of your DFT design process.

Using Mentor Graphics Documentation with Acrobat Exchange — describes how to set up and use the Mentor Graphics-supplied Acrobat Exchange with enhancement plugins for online viewing of Mentor Graphics PDF-based documentation and help. The manual contains procedures for using Mentor Graphics documentation, including setting up online manuals and help, opening documents, and using full-text searches. Also included are tips on using Exchange.
Command Line Syntax Conventions

The notational elements used in this manual for command line syntax are as follows:

**Bold**
A bolded font indicates a required argument.

[ ]
Square brackets enclose optional arguments (in command line syntax only). Do not enter the brackets.

**UPPercase**
Required command letters are in uppercase; you may omit lowercase letters when entering commands or literal arguments and you need not use uppercase. Command names and options are case insensitive. Commands usually follow the 3-2-1 rule: the first three letters of the first word, the first two letters of the second word, and the first letter of the third, fourth, etc. words.

**Italic**
An italic font indicates a user-supplied argument.

___
An underlined item indicates either the default argument or the default value of an argument.

{ }
Braces enclose arguments to show grouping. Do not enter the braces.

| The vertical bar indicates an either/or choice between items. Do not include the bar in the command.

…
An ellipsis follows an argument that may appear more than once. Do not include the ellipsis in commands.

You should enter literal text (that which is not in italics) exactly as shown.
Acronyms Used in This Manual

Below is an alphabetical listing of the acronyms used in this manual:

ASIC — Application Specific Integrated Circuit
ATE — Automatic Test Equipment
ATPG — Automatic Test Pattern Generation
AVI — ASIC Vector Interfaces
BIST — Built-In Self Test
BSDL — Boundary Scan Design Language
CUT — Circuit Under Test
DFT — Design-for-Test
DRC — Design Rules Checking
DUT — Device Under Test
GUI — Graphical User Interface
HDL — Hardware Description Language
JTAG — Joint Test Action Group
LFSR — Linear Feedback Shift Register
MCM — Multi-Chip Module
MISR — Multiple Input Signature Register
PRPG — Pseudo-Random Pattern Generator
SCOAP — Sandia Controllability Observability Analysis Program
Acronyms Used in This Manual

SFP — Single Fault Propagation

TAP — Test Access Port

TCK — Test Clock

TDI — Test Data Input

TDO — Test Data Output

TMS — Test Mode Select

TRST — Test Reset

VHDL — VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

WDB — Waveform DataBase
Chapter 1
Introduction

DFTAdvisor is Mentor Graphics scan identification and insertion tool. It is one of several tools in the Mentor Graphics Design-for-Test (DFT) tool suite. The following subsections list the features, inputs, and outputs of the tool. For information on using DFTAdvisor in the context of a DFT flow, refer to the “Inserting Internal Scan and Test Circuitry” chapter in the *Scan and ATPG Process Guide*.

Features

DFTAdvisor contains numerous features, including the following:

- Supports both full and partial-scan identification and insertion
- Supports the common scan methodologies, including Mux-scan, Clocked-scan, and LSSD
- Provides both automatic and manual scan identification capabilities allowing for an optimal partial scan solution
- Reads most standard gate-level netlists allowing you to use this product as a point tool in the overall DFT flow
- Contains a powerful design rules checker that helps ensure the most optimum ATPG downstream with the ATPG tools
- Automatically generates the scan setup dofile and the test procedure files for use later in the flow with the ATPG tools (FastScan and FlexTest)
- Can display a wide variety of useful information--from design and debugging information to statistical reports for the generated test set
Inputs and Outputs

DFTAdvisor utilizes the following inputs:

- **Design** - The supported netlist formats are EDIF, 2.0.0, Genie, VHDL, Verilog, TDL, and Spice.

- **Test Procedure File** - This file is only required if there is already preexisting scan circuitry in your design, and then this file defines the operation of that preexisting scan circuitry.

- **DFT Library** - This is the file that contains the model descriptions for all library cells used in your design along with the model descriptions for all the scan replacement cells.

- **Scan Setup File** - This is a set of commands that gives DFTAdvisor information on how to insert scan chains. You can also enter these commands interactively.

DFTAdvisor produces the following outputs:

- **Design** - This is the scan version of your design that is DFTAdvisor writes and saves as a netlist. The supported netlist formats are EDIF, 2.0.0, Genie, VHDL, Verilog, and TDL.

- **ATPG Setup Files** - These files include the test procedure file, which defines the operation of the scan circuitry in your design, and a dofile that sets up the design and scan circuitry information for ATPG. For more information on test procedure files, refer to “Test Procedure Files” in the *Scan and ATPG Process Guide*.
This Chapter contains descriptions of the DFTAdvisor commands. The subsections are named for the command they describe. For quick reference, the commands appear alphabetically with each beginning on a separate page.

Command Summary

Table 2-1 contains a summary of the commands described in this manual. The two columns that separate the command name and the description indicate the tools in which you can use the commands. The following tool acronyms are used in the table: DFTA = DFTAdvisor, DFTI = DFTInsight.

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTA</th>
<th>DFTI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Atpg Constraints</td>
<td>●</td>
<td></td>
<td>Specifies that the tool restrict all patterns it places into the internal pattern set according to the user-defined constraints.</td>
</tr>
<tr>
<td>Add Buffer Insertion</td>
<td>●</td>
<td></td>
<td>Specifies for DFTAdvisor to place buffer cells between the primary input of the specified test pin and the gates that it drives.</td>
</tr>
<tr>
<td>Add Cell Library</td>
<td>●</td>
<td></td>
<td>Specifies the EDIF library in which to place all or specified library models which are explicitly added by DFTAdvisor scan and test logic insertion process.</td>
</tr>
<tr>
<td>Add Cell Models</td>
<td>●</td>
<td></td>
<td>Specifies the name of a DFT library cell that DFTAdvisor can use with user-defined test points, system-generated test points, and system-generated test logic.</td>
</tr>
</tbody>
</table>
### Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTA</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Clock Groups</td>
<td>●</td>
<td>Specifies the grouping of scan cells controlled by different clocks onto one chain.</td>
</tr>
<tr>
<td>Add Clocks</td>
<td>●</td>
<td>Specifies the names and inactive states of the primary input pins that control the clocks in the design.</td>
</tr>
<tr>
<td>Add Display Instances</td>
<td>● ●</td>
<td>Adds the specified instances to the netlist for display.</td>
</tr>
<tr>
<td>Add Display Loop</td>
<td>● ●</td>
<td>Displays all the gates in a specified feedback path.</td>
</tr>
<tr>
<td>Add Display Path</td>
<td>● ●</td>
<td>Displays all the gates associated with the specified path.</td>
</tr>
<tr>
<td>Add Display Scanpath</td>
<td>● ●</td>
<td>Displays all the associated gates between two positions in a scan chain.</td>
</tr>
<tr>
<td>Add Faults</td>
<td>●</td>
<td>Adds faults into the current fault list.</td>
</tr>
<tr>
<td>Add LFSR Connections</td>
<td>●</td>
<td>Connects an external pin to a Linear Feedback Shift Register (LFSR).</td>
</tr>
<tr>
<td>Add LFSR Taps</td>
<td>●</td>
<td>Adds the tap configuration to a Linear Feedback Shift Register (LFSR).</td>
</tr>
<tr>
<td>Add LFSRs</td>
<td>●</td>
<td>Adds Linear Feedback Shift Registers (LFSRs) for use as Pseudo-Random Pattern Generators (PRPGs) or Multiple Input Signature Registers (MISRs).</td>
</tr>
<tr>
<td>Add Mapping Definition</td>
<td>●</td>
<td>Overrides the nonscan to scan model mapping defined by DFTAdvisor.</td>
</tr>
<tr>
<td>Add Mos Direction</td>
<td>●</td>
<td>Assigns the direction of a bi-directional MOS transistor.</td>
</tr>
<tr>
<td>Add Mtpi Controller</td>
<td>●</td>
<td>Creates a MTPI controller and connects it to the primary inputs.</td>
</tr>
<tr>
<td>Add Mtpi Output</td>
<td>●</td>
<td>Defines the values to be output by the controller.</td>
</tr>
<tr>
<td>Add Net Property</td>
<td>●</td>
<td>Defines the net in the Spice design and library as VDD or GND.</td>
</tr>
<tr>
<td>Add Nofaults</td>
<td>●</td>
<td>Places nofault settings either on a pin or on all pins of a specified instance or module.</td>
</tr>
</tbody>
</table>
### Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Nonscan Instances</td>
<td>Specifies for DFTAdvisor to ignore the specified instances, all instances controlled by the specified control pin, or all instances within the specified module, when identifying and inserting the required scan elements and test logic.</td>
</tr>
<tr>
<td>Add Nonscan Models</td>
<td>Instructs DFTAdvisor to ignore all instances of the specified sequential DFT library model when identifying and inserting the required scan elements and test logic into the design.</td>
</tr>
<tr>
<td>Add Notest Points</td>
<td>Adds circuit points to list for exclusion from testability insertion.</td>
</tr>
<tr>
<td>Add Output Masks</td>
<td>Instructs DFTAdvisor to mask, and optionally maintain a constant logic level on, the specified primary output pins during the scan identification analysis.</td>
</tr>
<tr>
<td>Add Pin Constraints</td>
<td>Specifies that DFTAdvisor hold the input pin at a constant state during the rules checking and loop cutting processes.</td>
</tr>
<tr>
<td>Add Pin Equivalences</td>
<td>Specifies to hold the specified primary input pins at a state either equal to or inverted in relationship to the state of another primary input pin during the rules checking.</td>
</tr>
<tr>
<td>Add Primary Inputs</td>
<td>Adds a primary input to the net.</td>
</tr>
<tr>
<td>Add Primary Outputs</td>
<td>Adds a primary output to the net.</td>
</tr>
<tr>
<td>Add Read Controls</td>
<td>Adds an off-state value to specified RAM read control lines.</td>
</tr>
<tr>
<td>Add Scan Chains</td>
<td>Specifies a name for a preexisting scan chain within the design.</td>
</tr>
<tr>
<td>Add Scan Groups</td>
<td>Adds one scan chain group to the system.</td>
</tr>
<tr>
<td>Add Scan Instances</td>
<td>Specifies that DFTAdvisor add the specified instance, all instances controlled by the specified control pin, or all instances within the specified module, to the scannable instance list.</td>
</tr>
</tbody>
</table>
Table 2-1. Command Summary  [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Scan Models</td>
<td>Specifies that DFTAdvisor is to flag every instance of the named DFT library model for inclusion into the identified scan list.</td>
</tr>
<tr>
<td>Add Scan Pins</td>
<td>Declares the name of a scan chain at the top-level module and assigns the corresponding scan input pin, scan output pin, and optionally, the scan clock pin that you wish to associate with that chain.</td>
</tr>
<tr>
<td>Add Seq_transparent</td>
<td>Constraints</td>
</tr>
<tr>
<td></td>
<td>Specifies the enable value of a clock enable that internally gates the clock input of a non-scan cell for sequential transparent scan identification.</td>
</tr>
<tr>
<td>Add Sub Chains</td>
<td>Specifies the name of a preexisting scan chain that exists entirely within a module or instance within a hierarchical design.</td>
</tr>
<tr>
<td>Add Test Points</td>
<td>Specifies explicitly where DFTAdvisor is to place a user-defined test point to improve the design’s testability either through better controllability or observability.</td>
</tr>
<tr>
<td>Add Tied Signals</td>
<td>Specifies for DFTAdvisor to hold the named floating objects (nets or pins) at the given state value.</td>
</tr>
<tr>
<td>Add Write Controls</td>
<td>Specifies the off-state value of the write control lines for RAMs.</td>
</tr>
<tr>
<td>Analyze Control Signals</td>
<td>Identifies and defines the primary inputs of control signals.</td>
</tr>
<tr>
<td>Analyze Drc Violation</td>
<td>Generates a netlist of the portion of the design involved with the specified rule violation number.</td>
</tr>
<tr>
<td>Analyze Input Control</td>
<td>Specifies for DFTAdvisor to calculate and display the effects of constraining primary input pins to an unknown value on those pins’ control capability.</td>
</tr>
<tr>
<td>Analyze Output Observe</td>
<td>Specifies for DFTAdvisor to calculate and display the effects on the observability of masked primary output pins.</td>
</tr>
<tr>
<td>Analyze Testability</td>
<td>Reports general scannability and testability information, along with calculating the controllability and observability values for gates.</td>
</tr>
<tr>
<td>Command</td>
<td>DFTA</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Close Schematic Viewer</td>
<td>●●</td>
</tr>
<tr>
<td>Delete Atpg Constraints</td>
<td>●</td>
</tr>
<tr>
<td>Delete Buffer Insertion</td>
<td>●</td>
</tr>
<tr>
<td>Delete Cell Models</td>
<td>●</td>
</tr>
<tr>
<td>Delete Clock Groups</td>
<td>●</td>
</tr>
<tr>
<td>Delete Clocks</td>
<td>●</td>
</tr>
<tr>
<td>Delete Display Instances</td>
<td>●●</td>
</tr>
<tr>
<td>Delete Faults</td>
<td>●</td>
</tr>
<tr>
<td>Delete LFSR Connections</td>
<td>●</td>
</tr>
<tr>
<td>Delete LFSR Taps</td>
<td>●</td>
</tr>
<tr>
<td>Delete LFSRs</td>
<td>●</td>
</tr>
<tr>
<td>Delete Mapping Definition</td>
<td>●</td>
</tr>
<tr>
<td>Delete Mos Direction</td>
<td>●</td>
</tr>
<tr>
<td>Delete Mtpi Controller</td>
<td>●</td>
</tr>
<tr>
<td>Delete Mtpi Output</td>
<td>●</td>
</tr>
<tr>
<td>Delete Net Property</td>
<td>●</td>
</tr>
</tbody>
</table>
Table 2-1. Command Summary  [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTA</th>
<th>DFTI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Nofaults</td>
<td>●</td>
<td></td>
<td>Removes the no-fault settings from either the specified pin or instance pathnames.</td>
</tr>
<tr>
<td>Delete Nonscan Instances</td>
<td>●</td>
<td></td>
<td>Removes the specified sequential instances from the non-scan instance list.</td>
</tr>
<tr>
<td>Delete Nonscan Models</td>
<td>●</td>
<td></td>
<td>Removes from the non-scan model list the specified sequential DFT library models.</td>
</tr>
<tr>
<td>Delete Notest Points</td>
<td>●</td>
<td></td>
<td>Removes the specified pins from the list of notest points which the tool cannot use for testability insertion.</td>
</tr>
<tr>
<td>Delete Output Masks</td>
<td>●</td>
<td></td>
<td>Removes the masking of the specified primary output pins.</td>
</tr>
<tr>
<td>Delete Pin Constraints</td>
<td>●</td>
<td></td>
<td>Removes the pin constraints from the specified primary input pins.</td>
</tr>
<tr>
<td>Delete Pin Equivalences</td>
<td>●</td>
<td></td>
<td>Removes the pin equivalence specifications for the designated primary input pins.</td>
</tr>
<tr>
<td>Delete Primary Inputs</td>
<td>●</td>
<td></td>
<td>Removes the specified primary inputs from the current netlist.</td>
</tr>
<tr>
<td>Delete Primary Outputs</td>
<td>●</td>
<td></td>
<td>Removes the specified primary outputs from the current netlist.</td>
</tr>
<tr>
<td>Delete Read Controls</td>
<td>●</td>
<td></td>
<td>Removes the read control line off-state definitions from the specified primary input pins.</td>
</tr>
<tr>
<td>Delete Scan Chains</td>
<td>●</td>
<td></td>
<td>Removes the specified scan chain definitions from the scan chain list.</td>
</tr>
<tr>
<td>Delete Scan Groups</td>
<td>●</td>
<td></td>
<td>Removes the specified scan chain group definitions from the scan group list.</td>
</tr>
<tr>
<td>Delete Scan Instances</td>
<td>●</td>
<td></td>
<td>Removes the specified sequential instances from the user-identified scan instance list.</td>
</tr>
<tr>
<td>Delete Scan Models</td>
<td>●</td>
<td></td>
<td>Removes the specified sequential models from the scan model list.</td>
</tr>
<tr>
<td>Delete Scan Pins</td>
<td>●</td>
<td></td>
<td>Removes any previously assigned scan input, output, and clock names from the specified scan chains.</td>
</tr>
</tbody>
</table>
### Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Seq_transparent Constraints</td>
<td>Removes the pin constraints from the specified DFT library model input pins.</td>
</tr>
<tr>
<td>Delete Sub Chains</td>
<td>Removes the definition of a preexisting scan sub-chain.</td>
</tr>
<tr>
<td>Delete Test Points</td>
<td>Remove the test point definitions at the specified locations.</td>
</tr>
<tr>
<td>Delete Tied Signals</td>
<td>Removes the assigned (tied) value from the specified floating nets or pins.</td>
</tr>
<tr>
<td>Delete Write Controls</td>
<td>Removes the RAM write control line off-state definitions from the specified primary input pins.</td>
</tr>
<tr>
<td>Dofile</td>
<td>Executes the commands contained within the specified file.</td>
</tr>
<tr>
<td>Exit</td>
<td>Terminates the current DFTAdvisor session.</td>
</tr>
<tr>
<td>Extract Subckts</td>
<td>Performs matching and conversion between the bi-directional MOS instance and the ATPG library model.</td>
</tr>
<tr>
<td>Flatten Subckt</td>
<td>Flattens the SUBCKT in the Spice design.</td>
</tr>
<tr>
<td>Help</td>
<td>Displays the usage syntax and system mode for the specified command.</td>
</tr>
<tr>
<td>Insert Scan Chains</td>
<td>Replaces and stitches each non-scan cell that DFTAdvisor has previously identified as a scan candidate with the corresponding scan cell.</td>
</tr>
<tr>
<td>Insert Test Logic</td>
<td>Inserts the test structures that you define into the netlist to increase the design’s testability.</td>
</tr>
<tr>
<td>Mark</td>
<td>Highlights the objects that you specify in the Schematic View window.</td>
</tr>
<tr>
<td>Open Schematic Viewer</td>
<td>Invokes the optional schematic viewing application, DFTInsight.</td>
</tr>
<tr>
<td>Read Subckts Library</td>
<td>Reads the specified Spice SUBCKT library.</td>
</tr>
<tr>
<td>Redo Display</td>
<td>Nullifies the schematic view effects of an Undo command.</td>
</tr>
</tbody>
</table>
### Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Atpg Constraints</td>
<td>Displays all the current ATPG state restrictions and the instance pins on which they reside.</td>
</tr>
<tr>
<td>Report Buffer Insertion</td>
<td>Displays a list of all the different scan test pins and the corresponding fanout limit.</td>
</tr>
<tr>
<td>Report Cell Models</td>
<td>Displays a list of either all cell models or the DFT library models associated with the specified cell type.</td>
</tr>
<tr>
<td>Report Clock Groups</td>
<td>Displays a list of all clock group definitions.</td>
</tr>
<tr>
<td>Report Clocks</td>
<td>Displays a list of all clock definitions.</td>
</tr>
<tr>
<td>Report Control Signals</td>
<td>Displays the rules checking results for the specified control signals.</td>
</tr>
<tr>
<td>Report Dft Check</td>
<td>Generates the scannability check results for non-scan instances.</td>
</tr>
<tr>
<td>Report Display Instances</td>
<td>Displays a textual report of the netlist information for either the gates or instances that you specify or for all the gates in the current schematic view display.</td>
</tr>
<tr>
<td>Report Drc Rules</td>
<td>Displays either a summary of all the Design Rule Check (DRC) violations or the data for a specific violation.</td>
</tr>
<tr>
<td>Report Environment</td>
<td>Displays the current values of all the “set” commands and the default names of the scan type pins.</td>
</tr>
<tr>
<td>Report Faults</td>
<td>Displays fault information from the current fault list.</td>
</tr>
<tr>
<td>Report Feedback Paths</td>
<td>Displays a textual report of the currently identified feedback paths.</td>
</tr>
<tr>
<td>Report Flatten Rules</td>
<td>Displays either a summary of all the flattening rule violations or the data for a specific violation.</td>
</tr>
<tr>
<td>Report Gates</td>
<td>Displays the netlist information for the specified gates.</td>
</tr>
<tr>
<td>Report LFSR Connections</td>
<td>Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.</td>
</tr>
<tr>
<td>Command</td>
<td>DFTA</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Report LFSRs</td>
<td>●</td>
</tr>
<tr>
<td>Report Loops</td>
<td>●</td>
</tr>
<tr>
<td>Report Mapping Definition</td>
<td>●</td>
</tr>
<tr>
<td>Report Mos Direction</td>
<td>●</td>
</tr>
<tr>
<td>Report Mtpi Controller</td>
<td>●</td>
</tr>
<tr>
<td>Report Net Properties</td>
<td>●</td>
</tr>
<tr>
<td>Report Nofaults</td>
<td>●</td>
</tr>
<tr>
<td>Report Nonscan Instances</td>
<td>●</td>
</tr>
<tr>
<td>Report Nonscan Models</td>
<td>●</td>
</tr>
<tr>
<td>Report Notest Points</td>
<td>●</td>
</tr>
<tr>
<td>Report Output Masks</td>
<td>●</td>
</tr>
<tr>
<td>Report Pin Constraints</td>
<td>●</td>
</tr>
<tr>
<td>Report Pin Equivalences</td>
<td>●</td>
</tr>
<tr>
<td>Report Primary Inputs</td>
<td>●</td>
</tr>
<tr>
<td>Report Primary Outputs</td>
<td>●</td>
</tr>
<tr>
<td>Report Read Controls</td>
<td>●</td>
</tr>
<tr>
<td>Report Scan Cells</td>
<td>●</td>
</tr>
</tbody>
</table>
## Table 2-1. Command Summary  [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Scan Chains</td>
<td>Displays a report on all the current scan chains.</td>
</tr>
<tr>
<td>Report Scan Groups</td>
<td>Displays a report on all the current scan chain groups.</td>
</tr>
<tr>
<td>Report Scan Identification</td>
<td>Displays a list of the scan instances which DFTAdvisor has identified or you have defined as scan cells.</td>
</tr>
<tr>
<td>Report Scan Instances</td>
<td>Displays the currently defined sequential scan instances.</td>
</tr>
<tr>
<td>Report Scan Models</td>
<td>Displays the sequential scan models currently in the scan model list.</td>
</tr>
<tr>
<td>Report Scan Pins</td>
<td>Displays all previously assigned scan input, output, and clock names.</td>
</tr>
<tr>
<td>Report Seq_transparent Constraints</td>
<td>Displays the seq_transparent constraints.</td>
</tr>
<tr>
<td>Report Statistics</td>
<td>Displays a detailed report of the design’s statistics.</td>
</tr>
<tr>
<td>Report Sub Chains</td>
<td>Generates and displays a report on the scan sub-chains.</td>
</tr>
<tr>
<td>Report Test Logic</td>
<td>Displays the test logic that DFTAdvisor added during the scan insertion process.</td>
</tr>
<tr>
<td>Report Test Points</td>
<td>Displays the test point specifications you created with Add Test Points command and any test points that you enabled DFTAdvisor to automatically identify.</td>
</tr>
<tr>
<td>Report Testability Analysis</td>
<td>Displays the results of the Analyze Testability command.</td>
</tr>
<tr>
<td>Report Tied Signals</td>
<td>Displays a list of the tied floating signals and pins.</td>
</tr>
<tr>
<td>Report Write Controls</td>
<td>Displays the currently defined write control lines and their off-states.</td>
</tr>
<tr>
<td>Reset State</td>
<td>Removes all instances from both the scan identification and test point identification lists that DFTAdvisor identified during a run.</td>
</tr>
<tr>
<td>Ripup Scan Chains</td>
<td>Removes the specified scan chains from the design.</td>
</tr>
</tbody>
</table>
Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>D F T A</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>● D F T I</td>
<td>Runs the scan or test point identification process in Dft mode and runs the fault simulation and signature calculation process in Bist mode.</td>
</tr>
<tr>
<td>Save Patterns</td>
<td>●</td>
<td>Saves the current BIST test pattern set to a file in the format that you specify.</td>
</tr>
<tr>
<td>Save Schematic</td>
<td>● ●</td>
<td>Saves the schematic currently displayed by DFTInsight.</td>
</tr>
<tr>
<td>Select Object</td>
<td>● ●</td>
<td>Selects the specified objects in the DFTInsight schematic view.</td>
</tr>
<tr>
<td>Set Bist Initialization</td>
<td>●</td>
<td>Specifies the scan chains input value which indicates the states of the scan cells before FastScan applies BIST patterns.</td>
</tr>
<tr>
<td>Set Capture Clock</td>
<td>●</td>
<td>Specifies the capture clock name for random pattern simulation.</td>
</tr>
<tr>
<td>Set Contention Check</td>
<td>●</td>
<td>Specifies whether DFTAdvisor checks the gate types that you determine for contention.</td>
</tr>
<tr>
<td>Set Control Threshold</td>
<td>●</td>
<td>Specifies the controllability value for simulation-based pseudorandom random pattern test point identification.</td>
</tr>
<tr>
<td>Set Dofile Abort</td>
<td>●</td>
<td>Lets you specify that the tool complete processing of all commands in a dofile regardless of an error detection.</td>
</tr>
<tr>
<td>Set Drc Handling</td>
<td>●</td>
<td>Specifies how DFTAdvisor globally handles design rule violations.</td>
</tr>
<tr>
<td>Set Fault Sampling</td>
<td>●</td>
<td>Specifies the fault sampling percentage for scan identification.</td>
</tr>
<tr>
<td>Set Flatten Handling</td>
<td>●</td>
<td>Specifies how DFTAdvisor globally handles flattening violations.</td>
</tr>
<tr>
<td>Set Gate Level</td>
<td>● ●</td>
<td>Specifies the hierarchical level of gate reporting and displaying.</td>
</tr>
<tr>
<td>Set Gate Report</td>
<td>● ●</td>
<td>Specifies the additional display information for the Report Gates command.</td>
</tr>
</tbody>
</table>
Table 2-1. Command Summary  [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTA</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Identification Model</td>
<td>●</td>
<td>Specifies the simulation model that DFTAdvisor uses to imitate the scan operation during the scan identification process.</td>
</tr>
<tr>
<td>Set Instancename Visibility</td>
<td>● ●</td>
<td>Specifies whether DFTInsight displays instance names immediately above each instance in the Schematic View area.</td>
</tr>
<tr>
<td>Set Internal Fault</td>
<td>●</td>
<td>Specifies whether the tool allows faults within or on the boundary of library models.</td>
</tr>
<tr>
<td>Set Io Insertion</td>
<td>●</td>
<td>Specifies whether to insert I/O buffers.</td>
</tr>
<tr>
<td>Set Latch Handling</td>
<td>●</td>
<td>Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.</td>
</tr>
<tr>
<td>Set Lockup Latch</td>
<td>●</td>
<td>Specifies for DFTAdvisor to insert latches between different clock domains to synchronize the clocks within a scan chain.</td>
</tr>
<tr>
<td>Set Logfile Handling</td>
<td>●</td>
<td>Specifies for DFTAdvisor to direct the transcript information to a file.</td>
</tr>
<tr>
<td>Set Loop Duplication</td>
<td>●</td>
<td>Specifies whether to include duplicate gates in feedback paths which are generated during the circuit flattening process.</td>
</tr>
<tr>
<td>Set Multiple Scan_enables</td>
<td>●</td>
<td>Specifies to create multiple scan_enables.</td>
</tr>
<tr>
<td>Set Net Resolution</td>
<td></td>
<td>Specifies the behavior of multi-driver nets.</td>
</tr>
<tr>
<td>Set Nonscan Handling</td>
<td>●</td>
<td>Specifies whether to check the nonscan instances for scannability.</td>
</tr>
<tr>
<td>Set Observe Threshold</td>
<td>●</td>
<td>Specifies the observability value for simulation-based test point identification.</td>
</tr>
<tr>
<td>Set Pattern Save</td>
<td>●</td>
<td>Enables the storing of BIST patterns.</td>
</tr>
<tr>
<td>Set Random Patterns</td>
<td>●</td>
<td>Specifies the number of random patterns DFTAdvisor uses for simulation.</td>
</tr>
</tbody>
</table>
## Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>D</th>
<th>F</th>
<th>T</th>
<th>I</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Scan Type</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies the scan style design.</td>
</tr>
<tr>
<td>Set Schematic Display</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>Changes the default schematic display environment settings for DFTInsight.</td>
</tr>
<tr>
<td>Set Screen Display</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies whether DFTAdvisor writes the transcript to the session window.</td>
</tr>
<tr>
<td>Set Sensitization Checking</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies whether DRC checking attempts to verify a suspected C3 rules violation.</td>
</tr>
<tr>
<td>Set Stability Check</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies how the tool checks the effect of applying the shift procedure on non-scan cells.</td>
</tr>
<tr>
<td>Set System Mode</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies the next system mode for the tool to enter.</td>
</tr>
<tr>
<td>Set Test Logic</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies which types of control lines DFTAdvisor makes controllable during the DFT rules checking.</td>
</tr>
<tr>
<td>Set Trace Report</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies whether the tool displays gates in the scan chain trace.</td>
</tr>
<tr>
<td>Set Zoom Factor</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>Specifies the scale factor that the zoom icons use in the DFTInsight Schematic View window.</td>
</tr>
<tr>
<td>Setup LFSRs</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Changes the shift_type and tap_type default setting for the Add LFSRs and Add LFSR Taps commands.</td>
</tr>
<tr>
<td>Setup Output Masks</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Sets the default mask for all output and bi-directional pins.</td>
</tr>
<tr>
<td>Setup Pin Constraints</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Sets the default pin constraint value for all input and bi-directional pins.</td>
</tr>
<tr>
<td>Setup Scan Identification</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Specifies the scan identification methodology and amount of scan that DFTAdvisor is to consider during the identification run.</td>
</tr>
<tr>
<td>Setup Scan Insertion</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Sets up the parameters for the Insert Scan Chains and Insert Test Logic commands.</td>
</tr>
<tr>
<td>Setup Scan Pins</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>Changes the scan-in or scan-out pin naming parameters to index or bus format.</td>
</tr>
</tbody>
</table>
## Table 2-1. Command Summary [continued]

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTA</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup Test_point Identification</td>
<td>●</td>
<td>Specifies the number of control and observe test points that DFTAdvisor flags during the identification run.</td>
</tr>
<tr>
<td>Setup Test_point Insertion</td>
<td>●</td>
<td>Specifies how DFTAdvisor configures the inputs for the system-defined control test points and the outputs for the system-defined observe test points.</td>
</tr>
<tr>
<td>Setup Tied Signals</td>
<td>●</td>
<td>Changes the default value for floating pins and floating nets which do not have assigned values.</td>
</tr>
<tr>
<td>System</td>
<td>●</td>
<td>Passes the specified command to the operating system for execution.</td>
</tr>
<tr>
<td>Undo Display</td>
<td>● ●</td>
<td>Restores the previous schematic view.</td>
</tr>
<tr>
<td>Unmark</td>
<td>● ●</td>
<td>Removes the highlighting from the specified objects in the Schematic View window of objects.</td>
</tr>
<tr>
<td>Unselect Object</td>
<td>● ●</td>
<td>Removes the specified objects from the selection list in the DFTInsight schematic view.</td>
</tr>
<tr>
<td>View</td>
<td>● ●</td>
<td>Displays the specified object in the DFTInsight Schematic View window.</td>
</tr>
<tr>
<td>View Area</td>
<td>● ●</td>
<td>Displays the specified area in the DFTInsight Schematic View window.</td>
</tr>
<tr>
<td>Write Atpg Setup</td>
<td>●</td>
<td>Writes the test procedure and the dofile for inserted scan chains to the specified files.</td>
</tr>
<tr>
<td>Write Bist Setup</td>
<td>●</td>
<td>Writes the top-level design interface and dofile used in LBISTArchitect.</td>
</tr>
<tr>
<td>Write Loops</td>
<td>●</td>
<td>Writes a list of all loops to the specified file.</td>
</tr>
<tr>
<td>Write Netlist</td>
<td>●</td>
<td>Writes the new netlist to the specified file.</td>
</tr>
<tr>
<td>Write Primary Inputs</td>
<td>●</td>
<td>Writes primary inputs to the specified file.</td>
</tr>
<tr>
<td>Write Primary Outputs</td>
<td>●</td>
<td>Writes primary outputs to the specified file.</td>
</tr>
</tbody>
</table>
The remaining pages in this chapter describe, in alphabetical order, the DFTAdvisor commands. Each command description begins on a new page. If you are looking for a command to accomplish a specific task but don’t know the command’s name, you can refer back to Table 2-1 on page 2-1 for help.

The notational conventions in use here are the same as those in use in other parts of the manual. Do not enter any of the special notational characters (such as, {}, [], or |) when typing the command. For a complete description of this manual’s notational conventions, refer to Command Line Syntax Conventions on page xvi in the About This Manual section.

You can use the line continuation character “\” when application commands extend beyond the end of a line. The line continuation character improves the readability of dofiles and helps with the command line entry of multiple-argument commands.

---

**Table 2-1. Command Summary [continued]**

<table>
<thead>
<tr>
<th>Command</th>
<th>DFTAI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Scan Identification</td>
<td>•</td>
<td>Writes a list of the scan instances which DFTAdvisor has identified or you have defined as scan cells.</td>
</tr>
<tr>
<td>Write Subchain Setup</td>
<td>•</td>
<td>Writes the appropriate Add Sub Chains commands to a file so that DFTAdvisor can understand the preexisting scan sub-chains at the top-level of this module.</td>
</tr>
<tr>
<td>Zoom In</td>
<td>• •</td>
<td>Enlarges the objects in the DFTInsight Schematic View window by reducing the displayed area.</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>• •</td>
<td>Reduces the objects in the DFTInsight Schematic View window by increasing the displayed area.</td>
</tr>
</tbody>
</table>
Add Atpg Constraints

Scope: Bist mode

Prerequisites: You can use this command only after the tool flattens the design to the simulation model, which happens when you first attempt to exit Setup mode.

Usage

ADD ATpg Constraints {0 | 1} pin_pathname...

Description

Specifies that the tool restrict all patterns it places into the internal pattern set according to the user-defined constraints.

The Add Atpg Constraints command allows you to change the ATPG constraints any time during the fault simulation process, affecting only the fault simulation that occurs after the constraint changes. The fault simulator rejects any subsequently simulated patterns that fail to meet the now current constraints.

If you previously used the multi-phase test point insertion method to insert test points into the design, you use this command in order to simulate the operation of the phase decoder while doing fault simulation.

When DFTAdvisor generates test patterns randomly, it does not have complete control over the highly automated process, which means that DFTAdvisor cannot ensure the use of the user-defined ATPG constraints. However, DFTAdvisor will reject non-conforming random patterns.

Arguments

• 0 | 1
  A literal that restricts the named pin to a low state, or a high state.

• pin_pathname
  A repeatable string that specifies the pathname to the pin on which you are placing the constraint.
Examples

The following example creates ATPG pin constraints:

```plaintext
set sys mod bist
add faults -all
del faults -untestable
set random patterns 2048
add atpg constr 0 /corecomp/core_i/phase_1
add atpg constr 0 /corecomp/core_i/phase_2
add atpg constr 1 /corecomp/core_i/phase_3
run
set random patterns 2048
del atpg constr /corecomp/core_i/phase_3
add atpg constr 0 /corecomp/core_i/phase_3
del atpg constr /corecomp/core_i/phase_2
add atpg constr 1 /corecomp/core_i/phase_2
run
set random patterns 2048
del atpg constr /corecomp/core_i/phase_2
add atpg constr 0 /corecomp/core_i/phase_2
del atpg constr /corecomp/core_i/phase_1
add atpg constr 1 /corecomp/core_i/phase_1
run
set random patterns 2047
del atpg constr /corecomp/core_i/phase_1
add atpg constr 0 /corecomp/core_i/phase_1
run
report lfsrs
```

Related Commands

- Delete Atpg Constraints
- Report Atpg Constraints
Add Buffer Insertion

Scope: All modes

Usage

ADD BUffer Insertion max_fanout test_pin... [-Model modelname]

Description

Specifies for DFTAdvisor to place buffer cells between the primary input of the specified test pin and the gates that it drives.

When DFTAdvisor inserts the scan circuitry into the design, the scan-related pins (enables and clocks) can fan out to drive multiple gates. When a pin has a large fanout, the pin’s increased load factor affects the quality of the pin’s output signal.

If you want to avoid signal degradation of a primary input scan pin, you can use the Add Buffer Insertion command. This sets the fanout limit on all buffers used to buffer the specified signal. The fanout limit is propagated down the buffer tree the tool inserts.

Arguments

- **max_fanout**

  A required integer that specifies the maximum number of gates the test_pin can drive before DFTAdvisor inserts buffers. The value must be greater than 1. By default, the tool assumes the test_pin can drive an infinite number of gates. This value overrides the default value set by the Add Cell Model command.

- **test_pin**

  A required repeatable literal that specifies the type of the primary input scan pin on which you want DFTAdvisor to monitor the maximum fanout. The following lists the default pin names for each type of scan pin. You can use the Setup Scan Insertion command to change the default names of the scan pins.

    **SEN** (scan enable; default name scan_en) — A literal that specifies the primary input pin that enables the scan chain.

    **SCLK** (scan clock; default name scan_clk) — A literal that specifies the primary input pin that clocks the scan data through the scan chain, which the clocked scan type uses.
**TEN** (test logic enable; default name test_en) — A literal that specifies the primary input pin that enables the operation of the test logic circuitry.

**TCLK** (test logic clock; default name test_clk) — A literal that specifies the primary input pin that clocks the values DFTAdvisor requires for proper functionality of the test logic.

**SMCLK** (master scan clock; default name scan_mclk) — A literal that specifies the primary input pin that clocks the scan data into the master scan elements of the scan chain when using the LSSD scan type.

**SSCLK** (slave scan clock; default name scan_sclk) — A literal that specifies the primary input pin that clocks the scan data into the slave scan elements of the scan chain when using the LSSD scan type.

**SET** (scan set; default name scan_set) — A literal that specifies the new scan set for the scan cells.

**RESET** (scan reset; default name scan_reset) — A literal that specifies the new scan reset for the scan cells.

- **-Model modelname**
  
  An optional switch and string pair that specifies the name of a buffer in the library that DFTAdvisor inserts when the scan pin reaches the maximum fanout. You must first identify the buffer with either the Add Cell Models command or with the cell_type library attribute. If you do not use the -Model switch, by default, the tool uses the first buffer model in the buffer cell model list (which you can obtain with the Report Cell Models command).

**Examples**

The following mux-DFF example explicitly specifies the buffer model to use and sets the maximum fanout for the scan enable pin itself.

You must first define the buf1a buffer model in the library using the `cell_type` library attribute with the value of “BIF”. The first command explicitly adds the buf2a cell to the buffer model list and defines its fanout to be 10. Next, the report shows the two buffers currently in the buffer model list. The last command specifies the maximum fanout of the scan enable pin and all buffers inserted to buffer the scan enable signal.
This example uses the -Model switch to specify the buf2a model. Without this switch, the tool would use the buf1a model, because it is the first in the buffer model list.

```bash
add cell models buf2a -type buf -max_fanout 10
report cell models
  BUF    : buf1a<infinity> buf2a<10>
add buffer insertion 5 sen -model buf2a
```

**Related Commands**

- Add Cell Models
- Delete Buffer Insertion
- Report Buffer Insertion
- Setup Scan Insertion
Add Cell Library

Scope: All modes

Prerequisites: This command is only useful when writing out an EDIF netlist.

Usage

ADD CELL Library library_name {{ -Model model_name } | -All}

Description

Specifies the EDIF library in which to place all or specified library models which are explicitly added by DFTAdvisor scan and test logic insertion process.

The Add Cell Library command lets you specify into which EDIF library to place the library models for inserted test logic. You can also specify an individual model of inserted test logic to place into the library.

Arguments

• library_name
  
  A required string that specifies the name of the EDIF library to create.

• {-Model model_name} | -All
  
  A required switch and string that lets you name the specific inserted test logic model or the entire library to place in the specified EDIF library.

Example

The following example specifies that all test logic inserted by DFTAdvisor is to be placed in the EDIF library “pad_lib”:

add cell library pad_lib -all

The following example specifies that if any test logic of model type “MUX21” was inserted by DFTAdvisor, the model cell definition is to be placed into the EDIF library “mux_lib”:

add cell library mux_lib -model MUX21
Add Cell Models

Scope: All modes

Usage

ADD CEll Models \texttt{dftlib\_model \{-Type \{INV | And \{Buf -Max\_fanout \textit{integer}\} \}
| OR | NAnd | NOR | Xor | INBuf | OUtbuf | \{Mux selector data0 data1\} | 
| Scancell \textit{clk} \textit{data} | \{DFf \textit{clk} \textit{data}\} | \{DLat enable data \{-Active \{High | Low\}\}\}} \}
| \{-Noinvert | -Invert\} \textit{output\_pin}\]

Description

Specifies the name of a DFT library cell that DFTAdvisor can use with user-defined test points, system-generated test points, and system-generated test logic.

Test logic is combinational circuitry that DFTAdvisor can add in front of sequential elements, memory elements, or enable lines of tri-state drivers. The purpose of adding test logic in front of sequential and memory elements is to ensure that the ATPG applications have the required control in order to make these devices scannable. The purpose of adding test logic in front of the enable lines of tri-state drivers is to avoid possible bus contention during the loading and unloading of the scan chains. The \textbf{Set Test Logic} command enables DFTAdvisor to check these elements for scannability and, when necessary, add the required test logic to ensure controllability of those elements.

If you enable test logic functionality with the Set Test Logic command, you need to specify the names of the DFT library cells that the tool is to use. If you do not specify the corresponding DFT library cells, then, when you issue the \textbf{Insert Scan Chains} command, DFTAdvisor does not know which cells to insert for the test logic. In this case, it displays an error message for each cell that does not have a corresponding DFT library cell.

The alternative to using the Add Cell Models command, is to use the \textit{cell\_type} attribute within the model definition of the DFT library itself.

If you are unsure of whether a particular design requires test logic, you can force DFTAdvisor to report the names of elements that require test logic. To use the reporting functionality, you enable the test logic functionality in Setup mode with the Set Test Logic command, enter Dft mode, and then issue the \textbf{Report Dft Check} command. This command lists the DFTAdvisor-identified pins that require test
logic for controllability. If the design requires the addition of test logic, you can then issue the Add Cell Models command.

If you add multiple cells of the same type with the Add Cell Models command, the tool uses the first added cell. You may want to add multiple cells of the same type if you are manually adding test points with the Add Test Points command because you can use different models of the same type at different test point locations.

For more information on the design library attribute, refer to the “Cell Type” subsection in “Design Library” section of the Design-for-Test Common Resources Manual.

Arguments

- **dftlib_model**
  A required string that specifies the name of a cell model in the DFT library that DFTAdvisor uses for test logic, buffer tree, lockup latch, or test point insertion.

- **-Type INV | And | {Buf -Max_fanout integer} | OR | NAND | NOR | Xor | INBuf | OUTbuf | {Mux selector data0 data1} | {Scancell clk data} | {DFf clk data} | {DLat enable data [-Active {High | Low}]}**
  A required switch and argument pair specifying the named *dftlib_model*’s type. The cell_model_type choices are as follows:
  
  **INV** — A literal that specifies a one-input inverter gate.
  
  **And** — A literal that specifies a two-input AND gate.
  
  **Buf** -Max_fanout integer — A literal with a switch and integer pair that specifies a one-input buffer gate with an optional buffer fanout limit.
  
  **OR** — A literal that specifies a two-input OR gate.
  
  **NAnd** — A literal that specifies a two-input NAND gate.
  
  **NOr** — A literal that specifies a two-input NOR gate.
  
  **Xor** — A literal that specifies an exclusive OR gate.
  
  **INBuf** — A literal that specifies a primary input buffer gate that DFTAdvisor inserts whenever the tool adds new input pins (such as the scan input or scan enable pins). It places the buffer between the primary input and the new pin.
**OUTbuf** — A literal that specifies a primary output buffer gate that DFTAdvisor inserts whenever the tool adds new output pins (such as the scan output pin). It places the buffer between the new pin and the primary output.

**Mux selector data0 data1** — A literal and three strings that specify a 2-1 multiplexer and the names of the selector pin and both data pins.

**Scancell clk data** — A literal and two strings that specify a mux-scan cell with four input pins (clock, data, scan in, and scan enable), clocked scan cell with four inputs (clock, data, scan clock, and scan enable), or LSSD scan cell with five inputs (clock, data, scan in, master clock, and slave clock). You must specify the name of the clock and data pins of the DFT library cell model. This option is meant to work in combination with the Add Test Points command.

**DFf clk data** — A literal and two strings that specify a D flip-flop with two input pins (clock and data). You must specify the names of the clock and data pins of the DFT library cell model. This option is meant to work in combination with the Add Test Points command.

**DLat enable data [-Active {High | Low}]** — A literal and two strings that specify a D latch with two input pins (enable and data). You must specify the names of the enable line and the data pin of the DFT library cell model. If you are defining this model for use with lockup latches, you may also include an optional switch with a literal. This switch specifies whether the enable input is active high or active low. This option is used with the Set Lockup Latch command. The default is active high.

- `{Noinvert | -Invert} output_pin`

An optional switch and string pair that you can use with values of the `cell_model_type` that are sequential elements. This switch specifies whether the `output_pin` has an inversion relationship with the data input of the given sequential element. By default, DFTAdvisor assumes no inversion relationship between the `output_pin` and the data input. If you do not explicitly specify an inversion switch, by default, DFTAdvisor uses the first `output_pin` value it identifies on a DFF, ScanCell, or DLAT model.
Examples

The following example shows a typical use of test logic, which involves the set, reset, and clock pins on sequential elements (flip-flops). DFTAdvisor can usually ensure controllability of sequential elements with model types of And, Or, and, Mux.

```
add clocks 0 clk
set test logic -set on -reset on -clock on
set system mode dft -force
report dft check
...
add cell models and2 -type and
add cell models or2 -type or
add cell models mux21h -type mux si a b
```

The following mux-DFF example adds the buf2a cell to the buffer model list, and then explicitly specifies the buffer model to use and sets the maximum fanout for the scan enable:

```
add cell models buf2a -type buff
report cell models
  BUF : buf1a  buf2a

add buffer insertion 5 sen -model buf2a
```

Related Commands

- Add Buffer Insertion
- Delete Cell Models
- Report Cell Models
- Set Io Insertion
- Set Lockup Latch
- Set Test Logic
- Setup Scan Identification
**Add Clock Groups**

Scope: Dft mode

Prerequisites: You must first define all the clocks with the Add Clocks command.

**Usage**

ADD CLock Groups *group_name clk_pin*... [-Tclk]

**Description**

Specifies the grouping of scan cells controlled by different clocks onto one chain. If you are going to merge scan cells controlled by multiple shift clocks into one scan chain, you can use the Add Clock Groups command to place the scan cells together that share the same shift clock.

DFTAdvisor groups scan cells controlled by the same clock in the chain. If you want to insert lockup latches between the different clock domains of the groups, you can use the Set Lockup Latch command. These latches synchronize the pulses to all the clock inputs of the scan cells within the same scan chain.

**Arguments**

- *group_name*
  
  A required string that specifies the name you want to assign to the list of clock pins that you provide with the *clk_pin* argument.

- *clk_pin*
  
  A required repeatable string that specifies the names of all the clocks that control the cells that you want to group together.

- -Tclk
  
  An optional switch specifying to include the test clock in the clock group. Because DFTAdvisor adds the clock signal during test structure insertion, you cannot specify the actual clock name here.

**Examples**

The following example lists the clocks in the current clock list, splits those clocks into two different groups, defines the latch DFTAdvisor is to use to synchronize
the different clocks, enables automatic lockup latch insertion, and then performs the scan and latch placement:

```
add clock 1 clk1 clk2
add clock 0 clk3 clk4 clk5 clk6
set system mode dft
...
add clock groups group1 clk1 clk3 clk4
add clock groups group2 clk2 clk5 clk6
add cell models dlat1a -type dlat enable data
add cell models inv -type inv
set lockup latch on
run
insert test logic -scan on -clock merge
```

This example causes DFTAdvisor to create two scan chains because there are two clock groups.

**Related Commands**

- Add Cell Models
- Set Lockup Latch
- Add Clocks
Add Clocks

Scope: Setup mode

Usage

ADD CLocks \textit{off\_state} \textit{primary\_input\_pin}...

Description

Specifies the names and inactive states of the primary input pins that control the clocks in the design.

You must declare all control signals (such as clocks, sets, and resets) and their corresponding off-states with the Add Clocks command before entering the Dft mode. Otherwise, instances that the design rules checker cannot completely control do not pass the scannability check. If an instance does not pass the scannability check, DFTAdvisor does not recognize it as a scannable instance, and cannot replace it with the corresponding scan cell.

Arguments

- \textit{off\_state}

  A required literal that specifies the pin value that cannot affect the output pin activity of the instance. For example, the off-state of an active low reset pin is 1 (high). For an edge-triggered control signal, the off–state is the value on the pin that results in the clock inputs being placed at the initial value of a capturing transition.

  The \textit{off\_state} choices are as follows:

  \begin{itemize}
  \item 0 — A literal specifying that the off-state value is 0.
  \item 1 — A literal specifying that the off-state value is 1.
  \end{itemize}

- \textit{primary\_input\_pin}

  A required repeatable string that lists the primary input pins that you want controlling the output pins of an instance. The list of primary input pins must all have the same \textit{off\_state}.

  If you declare a control pin with the Add Clocks command, DFTAdvisor also automatically declares all pins that are equal to that pin as control pins, by looking at the arguments of any Add Pin Equivalences commands.
Examples

The following example first lists the primary inputs of the design, which, in this case, is simply a D flip-flop. The next two commands declare the preset, clear, and clock pins to be clocks, which means they have the ability to control the states on the output pins of that instance.

```
report primary inputs
SYSTEM: /CLK_INPUT
SYSTEM: /D_INPUT
SYSTEM: /PRE_INPUT
SYSTEM: /CLR_INPUT
```

```
add clock 1 /pre_input /clr_input
add clock 0 /clk_input
```

Related Commands

- Add Clock Groups
- Report Clocks
- Delete Clocks
Add Display Instances

Tools Supported: DFTAdvisor and DFTInsight

Scope: Bist and Dft mode

Usage

ADD Display Instances \{gate_id\# [\-I input_pin_id \-O output_pin_id]] | pin_pathname | instance_name}... [-Forward | -Backward] [stopping_point]

DFTInsight Menu Paths:
- Display > Additions: Named Instances
- Display > Back Trace >...
- Display > Forward Trace >...

Description

Adds the specified instances to the netlist for display.

The Add Display Instances command creates a netlist containing the gates that you specify. If you already have DFTInsight invoked, the viewer automatically displays the graphical representation of the netlist and also marks key instances in the schematic view. Otherwise (if licensed), DFTInsight is automatically invoked on the netlist.

Arguments

The following lists the three methods for naming the objects that you want DFTInsight to display. You can use any number of the three argument choices, in any order.

- \textbf{gate_id\#} \-I input_pin_id \-O output_pin_id

A repeatable integer with an optional switch and number pair that specifies the gates that DFTInsight displays. The value of the \textit{gate_id\#} argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

You can optionally specify an input or output pin identification number for each gate by appending one of the following switch and number pairs to the \textit{gate_id\#}: 

- \textbf{pin_pathname}
- \textbf{instance_name}
gate_id\# -I input_pin_id\# — A gate identification number with an optionally appended switch and number pair that specifies the input pin identification number.

The tool assigns the input pins their identification numbers beginning with the upper pins and moving to the lower pins, starting with the number zero. DFTInsight then displays the gates that connect to the specified input pin of the given gate_id\#.

gate_id\# -O output_pin_id\# — A gate identification number with an optionally appended switch and number pair that specifies the output pin identification number.

The tool assigns the output pins their identification numbers beginning with the upper pins and moving to the lower pins, starting with the number zero. DFTInsight then displays the gates that connect to the specified output pin of the given gate_id\#.

- **pin_pathname**
  A repeatable string that specifies the name of a top-level pin within the design. DFTInsight displays the gate for that pin_pathname.

- **instance_name**
  A repeatable string that specifies the name of a top-level instance within the design. DFTInsight displays the gate for that instance_name.

- **-Forward**
  An optional switch specifying that the trace from the given objects is forward, towards the primary output pins. This is the command’s default.

  If you do not explicitly specify a stopping_point switch in combination with this switch, the command default is for the forward trace to include only one level of gates.

- **-Backward**
  An optional switch specifying that the trace from the given objects is backward, towards the primary input pins.

  If you do not explicitly specify a stopping_point switch in combination with this switch, the command default is for the backward trace to include only one level of gates.
• **stopping_point**

An optional switch argument that specifies the last gate that you want DFTInsight to include in the display. The following information describes the stopping_point choices, from which you can select only one:

- **-Level number** — A switch and integer pair that specifies for DFTInsight to stop the trace after it reaches the given number of connected gates. If you do not use one of the stopping_point arguments with the command, the default is -Level 1. You can use this switch in combination with either the -Forward or -Backward switch.

- **-Cone** — A switch that specifies for DFTInsight to stop the trace after it reaches all the gates in a cone of a clock. A cone is bound by tie gates, state elements, primary inputs, and primary outputs. This switch requires that you specify the direction in which DFTInsight performs the trace by using either the -Forward or -Backward switch.

- **-End_point** — A switch that specifies for DFTInsight to continue the trace until it reaches either a primary input, primary output, or a tie gate. This switch requires that you specify the direction in which DFTInsight performs the trace by using either the -Forward or -Backward switch.

- **-Decision_point** — A switch that specifies for DFTInsight to continue the trace until it reaches a multiple-input gate. The trace includes all the inputs of the multiple-input gate, but stops after that point. This switch requires that you specify the direction in which DFTInsight performs the trace by using either the -Forward or -Backward switch.

**Examples**

The following paragraphs provide examples that use the Add Display Instances command to display various gates.

The first example invokes DFTInsight, then displays a single gate by specifying the gate identification number (51).

    open schematic viewer
    add display instances 51

The next example specifies that the tool additionally display the next three levels of fanout gates from the number one input of gate 51. The command displays the
gate that feeds the number one input (first level), all the fanout gates from gate 51 (second level), plus all the gates that fanout from those gates (third level).

\textbf{add display instances 51 -I 1 -F -Level 3}

The final example clears the schematic display of all gates, then creates a new display that shows the associated gate for the specified instance, along with a backtracking of all the gates until the trace reaches either a primary input or tie gate.

\textbf{delete display instances -all}
\textbf{add display instances i_7_16 -b -end_point}

\textbf{Related Commands}

Delete Display Instances  
Report Display Instances  
Open Schematic Viewer
Add Display Loop

Tools Supported: DFTAdvisor and DFTInsight

Scope: Bist and Dft modes

Prerequisites: You can use this command only after the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode.

Usage

ADD Display Loop pin_pathname | feedback_id... | -All

DFTInsight Menu Path:
  Display > Additions: Loop

Description

Displays all the gates in a specified feedback path.

The DFTAdvisor circuit learning process provides an identification number and a list of gates for each feedback path. The Add Display Loop command creates a netlist containing specified feedback paths. By default, the gate lists include any duplicated gates. You can suppress duplicated gates by using the Set Loop Duplication command prior to initiating the circuit learning process.

The Add Display Loop command allows you to specify a feedback path by its identification number. You can display a list of all the feedback path identification numbers by using the Report Feedback Paths command.

If you already have invoked DFTInsight on a flattened design, the viewer automatically displays the graphical representation of the netlist and also marks key instances in the schematic view. Otherwise (if licensed), DFTInsight is automatically invoked on the netlist.

Arguments

- **pin_pathname**
  
  A string that specifies the pin_pathname of a feedback path gate. When you specify a gate pin name, DFTInsight displays the complete feedback path in which the gate resides.
**feedback_id#**
A repeatable integer that specifies the identification number of the feedback path whose gates you want DFTInsight to display.

**-All**
A switch specifying that DFTInsight display the gates for all of the feedback paths.

**Examples**
The following example invokes the optional schematic viewing application, leaves the Setup mode (thereby flattening the design and performing the learning process), displays the identification numbers of any learned feedback paths, and then schematically displays one of the feedback paths:

```
open schematic viewer
set system mode dft
report feedback paths
Loop#=0, feedback_buffer=26, #gates_in_network=5
   INV /I_956__I_582/ (51)
   PBUS /I_956__I_582/N1/ (96)
   ZVAL /I_956__I_582/N1/ (101)
   INV /I_956__I_582/ (106)
   TIEX /I_956__I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
   INV /I_962__I_582/ (52)
   PBUS /I_962__I_582/N1/ (95)
   ZVAL /I_962__I_582/N1/ (100)
   INV /I_962__I_582/ (105)
   TIEX /I_962__I_582/ (27)
add display loop 1
```

**Related Commands**

- Report Feedback Paths
- Set Loop Duplication
Add Display Path

Tools Supported: DFTAdvisor and DFTInsight

Scope: Bist and Dft modes

Prerequisites: This command can only operate on the flattened design. The design flattening happens when you first attempt to exit Setup mode.

Usage

ADD Display Path \{gate_id_begin# | instance_name_begin\} [gate_id_end# | instance_name_end] [-Noblock]

DFTInsight Menu Path:
Display > Additions: Delay Path

Description

Displays all the gates associated with the specified path.

The Add Display Path command creates a netlist containing the named path. If you already have invoked DFTInsight on a flattened design, the viewer automatically displays the graphical representation of the netlist and also marks key instances in the schematic view. Otherwise (if licensed), DFTInsight is automatically invoked on the netlist.

You specify a particular path by indicating the beginning gate or instance and the end gate or instance of the path or by just indicating the beginning gate or instance if the path is a loop. If the tool cannot identify a path or a loop, then it displays an error message. State elements and tie gates block the path unless you specify the -Noblock switch.

Arguments

- \textit{gate_id_begin#}

An integer specifying the gate identification number of the first gate in the path that you want the DFTInsight schematic viewer to display. The value of the \textit{gate_id_begin#} argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.
If you pair this argument with a `gate_id_end#` argument, the command displays all the gates between `gate_id_begin#` and `gate_id_end#`.

If you only specify the `gate_id_begin#`, then the tool assumes the path is a loop. If the tool does not find a loop, then it displays an error message.

- **instance_name_begin**
  A string specifying the name of the first gate instance in the path you want to display in the DFTInsight schematic viewer.
  If you pair this argument with an `instance_name_end` argument, the command displays all the gates between `instance_name_begin` and `instance_name_end`.
  If you only specify the `instance_name_begin`, then the tool assumes the path is a loop. If the tool does not find a loop, it displays an error message.

- **instance_name_end**
  An optional string specifying the name of the last gate instance in the path that you want the DFTInsight schematic viewer to display. You can only pair this argument with the `instance_name_begin` argument.

- **gate_id_end#**
  An optional integer specifying the gate identification number of the last gate in the path that you want the DFTInsight schematic viewer to display. The value of the `gate_id_end#` argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.
  You can only pair this argument with the `gate_id_begin#` argument.

- **-Noblock**
  An optional switch that causes the tool to not allow state elements and tie gates to block the path.

### Examples

The following example invokes DFTInsight, then displays a custom gate path by specifying the first and last gate identification numbers in the path (51 and 65):

```plaintext
open schematic viewer
add display path 51 65
```
Add Display Scanpath

Tools Supported: DFTAdvisor and DFTInsight

Scope: Bist and Dft modes

Prerequisites: This command can only operate on the flattened design. The design flattening happens when you first attempt to exit Setup mode.

Usage

ADD Display Scanpath chain_name [SCI | begin_cell_position] [SCO | end_cell_position]

DFTInsight Menu Path:
Display > Additions: ScanPath

Description

Displays all the associated gates between two positions in a scan chain.

The Add Display Scanpath command creates a netlist containing either all the gates or a subset of gates in a scan chain. If you invoke DFTInsight on a flattened design, the viewer automatically displays the graphical representation of the netlist and also marks key instances in the schematic view. Otherwise, you must invoke DFTInsight by issuing the Open Schematic Viewer command to see the resulting netlist.

You can specify a particular subset of a scan chain by indicating the beginning cell position and the ending cell position within the scan chain. By default, the command uses the scan chain primary input (SCI) and the scan chain primary output (SCO).

You can display a list of all the currently defined scan chains by using the Report Scan Chains command.

Arguments

- chain_name
  
  A required string specifying the name of the scan chain that you want to display in the DFTInsight schematic viewer. The scan chain must be a currently-defined scan chain.
• **SCI**
  An optional literal that causes DFTI to begin the scan chain display with the primary input gate of the `chain_name`. The primary input gate connects to the scan chain cell whose cell number equals the total number of scan cells minus one. This is the default.

• **begin_cell_position**
  An optional integer that specifies the position in a scan cell of the first cell that you want to display. The cell position must be an integer where 0 is the scan cell closest to the scan-out pin. You can determine the position of a cell within a scan chain by using the Report Scan Cells command.

• **SCO**
  An optional literal that causes DFTI to end the scan chain display with the primary output gate of the `chain_name`. The primary output gate connects to the scan chain cell whose cell number is 0. This is the default.

• **end_cell_position**
  An optional integer that specifies the position in a scan cell of the last cell that you want to display. The cell position must be an integer where 0 is the scan cell closest to the scan-out pin. You can determine the position of a cell within a scan chain by using the Report Scan Cells command.

**Examples**

The following example invokes DFTInsight, then displays a portion of a scan chain from its primary input gate to its eighth cell from the scan chain output:

```
open schematic viewer
add display scanpath chain1 sci 8
```

The next example displays the logic between the last scan cell and the scan chain output pin:

```
add display scanpath chain1 0 sco
```

**Related Commands**

Add Scan Chains  Report Scan Chains
Delete Scan Chains
Add Faults

Scope: Bist mode

Usage

ADD Faults {object_pathname... | -All} [-Stuck_at {01 | 0 | 1}]

Description

Adds faults into the current fault list.

The Add Faults command adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected. When you enter the Setup mode, DFTAdvisor deletes all faults from the current fault list. The tool only adds one instance of any given fault, ignoring any duplicate faults.

Arguments

- **object_pathname**
  A repeatable string specifying pins or instances whose faults the tool adds to the current fault list.

- **-All**
  A switch specifying that the tool add all of the faults on all model, netlist primitive, and top module pins.

- **-Stuck_at 01 | 0 | 1**
  An optional switch and literal pair that specifies which stuck-at faults to add to the fault list. The stuck-at values are as follows:

  - **01** — A literal specifying that the tool add both the “stuck-at-0” and “stuck-at-1” faults. This is the default.
  - **0** — A literal specifying that the tool add only the “stuck-at-0” faults.
  - **1** — A literal specifying that the tool add only the “stuck-at-1” faults.
Examples

The following example adds all faults to the circuit so that you can run the fault simulation and signature calculation process:

```bash
set system mode bist
add faults -all
run
```

Related Commands

Delete Faults  Report Faults
Add LFSR Connections

Scope: Setup mode
Prerequisites: This command intended for an LBISTArchitect design flow.

Usage
ADD LFsr Connections primary_pin lfsr_name position...

Description
Connects an external pin to a Linear Feedback Shift Register (LFSR).
The Add LFSR Connections command establishes a connection between the LFSR and the appropriate primary pins of the design.

LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. DFTAdvisor assumes that the output of the 0 bit position connects to the input of the highest bit position. If you select multiple bits of a Pseudo-Random Pattern Generator (PRPG) for the position argument, the tool assumes they are all exclusive-ORed together to create the value for the pin.

If you determine that multiple primary_pins must connect to a bit position of a Multiple Input Signature Register (MISR), you must issue a separate Add LFSR Connections command for each pin. The tool assumes the pins are all exclusive-ORed together to create the value for the next MISR input.

DFTAdvisor also assumes that the physical placement of any MISR connection is after the corresponding tapping point as shown in Figure 2-1.

Figure 2-1. MISR pin placement
You can use the Report LFSRs command to display all the LFSRs with their current values and tap positions.

**Arguments**

- **primary_pin**
  A required string that specifies the name of the design or primary input pin that you want to connect to the LFSR specified by `lfsr_name`.

- **lfsr_name**
  A required string that specifies the name of the LFSR to which you want to connect the `primary_pin`.

- **position**
  A required repeatable integer that specifies the bit positions of the `lfsr_name` at whose outputs you wish to place connections. A bit position is an integer number, where 0 indicates the least significant bit position. The tool assumes the output of the 0 bit position connects to the input of the highest bit position.

**Examples**

The following example connects an LFSR to a scan-in pin and another LFSR to a scan-out pin:

```
add lfsrs lfsr1 prpg 5 10 -serial -in
add lfsrs misr1 misr 5 15 -both -out
add lfsr taps lfsr1 1 3
add lfsr taps misr1 1 2
add lfsr connections scan_in.1 lfsr1 2
add lfsr connections scan_out.0 misr1 3
```

**Related Commands**

- Add LFSRs
- Add LFSR Taps
- Delete LFSR Connections
- Report LFSR Connections
Add LFSR Taps

Scope: Setup mode
Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

ADD LFsr Taps  lfsr_name position...

Description

Adds the tap configuration to a Linear Feedback Shift Register (LFSR).

The Add LFSR Taps command sets the tap configuration of an LFSR. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. DFTAdvisor assumes the output of the 0 bit position connects to the selected tap points and that the 0 bit position cannot itself be a tap point. You use this command primarily for implementing Built-In Self-Test (BIST) circuitry.

You can use the Report LFSRs command to display all the LFSRs with their current values and tap positions. You can change the default setting of the tap_type switches by using the Setup LFSRs command.

Arguments

•  lfsr_name
  A required string that specifies the name of the LFSR on which you want to place the taps.

•  position
  A required repeatable integer that specifies the bit positions of the lfsr_name at whose outputs you wish to place the taps.

Examples

The following example places taps on the newly added LFSRs:

```
add lfsrs lfsr1 prpg 5 10 -serial -in
add lfsrs misr1 misr 5 15 -both -out
add lfsr taps lfsr1 1 3
add lfsr taps misr1 1 2
```
Related Commands

Add LFSRs
Delete LFSR Taps
Report LFSRs
Setup LFSRs
Add LFSRs
Scope: Setup mode
Prerequisites: This command intended for an LBISTArchitect design flow.

Usage
ADD LFsrs lfsr_name {Prpg | Misr} length seed [-Both | -Serial | -Parallel]
[ -Out | -In]

Description
Adds Linear Feedback Shift Registers (LFSRs) for use as Pseudo-Random Pattern Generators (PRPGs) or Multiple Input Signature Registers (MISRs).

The Add LFSRs command defines LFSRs, which the design uses as PRPGs, to create pseudo-random values for the Built-In Self-Test (BIST) patterns or as MISRs to compact responses.

You specify the LFSR’s shift technique by using one of the following shift_type switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following tap_type switches: -Out or -In. You can change the default setting of the shift_type and tap_type switches by using the Setup LFSRs command.

Arguments
- **lfsr_name**
  A required string that specifies the name that you want to assign to the LFSR.
- **Prpg**
  A literal that indicates the LFSR functions as a PRPG.
- **Misr**
  A literal that indicates the LFSR functions as a MISR.
- **length**
  A required integer, greater than 1, specifying the number of bits in the LFSR.
• **seed**
  A required, right-justified, hexadecimal number, greater than 0, specifying the initial state of the LFSR.

• The following lists the three shift_type switches, of which you can choose only one.
  - **Both** — An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default unless you change the default with the Setup LFSRs command.
  - **Serial** — An optional switch specifying that the LFSR shifts serially the number of times equal to the length of the longest scan chain for each scan pattern.
  - **Parallel** — An optional switch specifying that the LFSR parallel shifts once for each scan pattern.

• **-Out | -In**
  An optional switch specifying that the exclusive-OR taps reside outside or inside the register path. -Out is the default unless you change the default with the Setup LFSRs command.

**Examples**

The following example defines an LFSR to be a PRPG and another LFSR to be a MISR:

```
add lfsrs lfsr1 prpg 5 10 -serial -in
add lfsrs misr1 misr 5 15 -both -out
add lfsr taps lfsr1 1 3
add lfsr taps misr1 1 2
```

**Related Commands**

- Add LFSR Taps
- Add LFSR Connections
- Delete LFSRs
- Report LFSRs
- Setup LFSRs
Add Mapping Definition

Scope: Setup and DFT modes

Prerequisites: You can only override the mapping for scan models of the same scan type. For example, a mux-DFF scan model can only be replaced by another mux-DFF scan model.

Usage

ADD MApping Definition object_name [-Instance | -Module] [-Nonscan_model nonscan_model_name] [-Scan_model scan_model_name] [-Output scan_output_pin_name]

Description

 Overrides the nonscan to scan model mapping defined by DFTAdvisor.

The Add Mapping Definition command defines the mapping of nonscan models to scan models. You can change the scan model for an individual instance, all instances under a hierarchical instance, all instances in all occurrences of a module in the design, or all occurrences of the model in the entire design. Additionally, you can change the scan output pin of the scan model in the same manner.

Note

The intent of this command is to change the existing nonscan to scan model mapping, as defined in the library, and not to define the mapping.

For additional information on scan cell and scan output mapping, refer to “Defining Scan Cell and Scan Output Mapping” in the Scan and ATPG Process Guide.

Arguments

• object_name

A required string that specifies the name of the nonscan model you want to map to a different scan model. You can also specify an instance, hierarchical instance, module, or scan model.
o If this argument is the name of an instance or hierarchical instance, the -Instance switch is required and the model must be specified with the -Nonscan_model switch or -Scan_model switch.

o If this argument is the name of a module, then the -Module switch is required and the model must be specified with the -Nonscan_model or -Scan_model switch.

o If this argument is a scan model, then the -Output switch is required. Since you specified a scan model, you can only define the scan output pin mapping.

• -Instance | -Module
  An optional switch that specifies the type of the object_name argument. If neither switch is specified, the object_name is a model (the default).

  o If you specify -Instance and the instance is primitive, then only the named instance has its mapping changed.

  o If you specify -Instance and the instance is hierarchical, then all instances under that instance matching the -Nonscan_model or (for output mapping) matching the -Scan_model have their mapping changed.

  o If you specify -Module, then for all occurrences of that module, all instances within that module that match the -Nonscan_model or (for output mapping) matching the -Scan_model have their mapping changed.

• -Nonscan_model nonscan_model_name
  A switch and string pair that specifies the name of the nonscan model that you want to change the mapping. This argument is required only if you specify -Instance or -Module switch, otherwise, you can specify the nonscan model in the object_name argument.

• -Scan_model scan_model_name
  A switch and string pair that specifies the name of the scan model that you want to use for the specified nonscan model. This argument is required except
when you are only changing the mapping of the scan output pin and specify the scan model in the object_name argument.

- **-Output scan_output_pin_name**

An optional switch and string pair that specifies the name of the scan output pin to use instead of the DFTAdvisor defined scan output pin. The port must have been declared as a scan out port in the scan_definition section of the scan cell.

### Examples

The following example maps the fd1 nonscan model to the fd1s scan model for all occurrences of the model in the design:

```
add mapping definition fd1 -scan_model fd1s
```

The following example maps the fd1 nonscan model to the fd1s scan model and changes the scan output pin to “qn” for all occurrences of the model in the design:

```
add mapping definition fd1 -scan_model fd1s -output qn
```

The first command in the following example maps the fd1 nonscan model to the fd1s scan model for all matching instances in the “counter” module and for all occurrences of that module in the design. The second command maps the fd1 nonscan model to the fd1s2 scan model and changes the scan output pin to “qn” for all matching instances under the hierarchical instance “/top/counter1”. Note that counter1 is an instance of the module counter changed in the first command.

```
add mapping definition counter -module -nonscan_model fd1 -scan_model fd1s
add mapping definition /top/counter1 -instance -nonscan_model fd1 -scan_model fd1s2 -output qn
```

The following example changes the scan output pin to “qn” for all occurrences of the fd1s scan model in the design:

```
add mapping definition fd1s -output qn
```

### Related Commands

- Delete Mapping Definition
- Report Mapping Definition
Add Mos Direction

Scope: Setup mode
Prerequisites: This command can only operate on a Spice design.

Usage
ADD MOs Direction subckt_name instance_name source_port drain_port

Description
Assigns the direction of a bi-directional MOS transistor.
The Add Mos Direction command sets the direction of a bi-directional transistor in the Spice design or library. The direction is from SOURCE to DRAIN port.

Arguments
• subckt_name
  A required string that specifies the name of the SUBCKT that contains the instance for which you are setting the direction.
• instance_name
  A required string that specifies the name of the instance within the SUBCKT for which you are setting the direction.
• source_port
  A required string that specifies the name of the SOURCE port.
• drain_port
  A required string that specifies the name of the DRAIN port.

Examples
The following example assigns the direction of the instance (K5) bi-directional MOS transistor of the subckt FADD2 from the port IN0 to the port IN1:
add mos direction FADD2 K5 IN0 IN1
Related Commands

- Extract Subckts
- Delete Mos Direction
- Report Mos Direction
Add MTpi Controller

Scope: Setup mode

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

ADD MTpi Controller controller_name primary_input...

Description

Creates a MTPI controller and connects it to the primary inputs.

The Add MTpi Controller command creates a MTPI controller and connects it to the specified primary inputs of the design.

Arguments

- controller_name
  A string that specifies the name of the MTPI controller to create.

- primary_input
  A repeatable string that specifies the names of the primary input of the design. These inputs may not be clocks, RAM control signals, or LFSRs functioning as PRPGs. The order that the primary inputs are entered is used to map to the MTPI controller output pins.

Examples

The following example defines one controller with three primary inputs. It also defines the outputs of the controller.

```
add mtpi controller controller1 /corecomp/core_i/phase_1 /corecomp/core_i/phase_2 /corecomp/core_i/phase_3
add mtpi output controller1 0 001
add mtpi output controller1 2 010
add mtpi output controller1 9 100
add mtpi output controller1 18 000
```
Add Mtpi Controller

Related Commands

<table>
<thead>
<tr>
<th>Add Mtpi Output</th>
<th>Report Mtpi Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Mtpi Controller</td>
<td>Set Bist Initialization</td>
</tr>
<tr>
<td>Delete Mtpi Output</td>
<td></td>
</tr>
</tbody>
</table>
Add Mtpi Output

Scope: All modes

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

ADD MTpi Output controller_name {cycle_number output_value}...

Description

Defines the values to be output by the controller.

The Add Mtpi Output command defines the values to be output by the controller when the pattern count reaches the specified cycle number.

Arguments

• controller_name
  A string that specifies the name of the MTPI controller for which you are defining the output values.

• cycle_number output_value
  A pair of repeatable strings that specify the cycle_number at which the accompanying output_value is placed at the output. The output_value is a string of binary digits, where the length of the string must match the number of primary inputs attached to the MTPI controller. Each bit in the string is mapped to the MTPI controller primary input connection in the same order as the pins were defined with the Add Mtpi Controller command.

Examples

The following example defines one controller with three primary inputs. It also defines the outputs of the controller. The output values are defined for cycle number 0, 2, 9, and 18. Notice that the output value is three digits, which matches the number of primary inputs.
Add Mtpi Output

Command Dictionary

add mtpi controller controller1 /corecomp/core_i/phase_1
            /corecomp/core_i/phase_2 /corecomp/core_i/phase_3
add mtpi output controller1 0 001
add mtpi output controller1 2 010
add mtpi output controller1 9 100
add mtpi output controller1 18 000

Related Commands

Add Mtpi Controller
Delete Mtpi Controller
Delete Mtpi Output

Report Mtpi Controller
Set Bist Initialization
Add Net Property

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

ADD NET Property net_name [-VDD | -GND]

Description

Defines the net in the Spice design and library as VDD or GND.

The Add Net Property command defines the specified net as VDD or GND in the Spice design and Spice library by adding a property.

Arguments

- **net_name**
  A required string that specifies the name of the net which you want to define as VDD or GND.

- **-VDD | -GND**
  A required switch that specifies whether the net is VDD or GND.

Examples

The following example defines the ZGND net as GND in the loaded Spice design and Spice library.

    add net property ZGND -gnd

Related Commands

Delete Net Property  Report Net Properties
Add Nofaults

Scope: Setup mode

Usage

ADD NOfaults *pathname* [-Instance | -Module] [-Stuck_at {01 | 0 | 1}] [-Keep_boundary]

Description

Places nofault settings either on a pin or on all pins of a specified instance or module.

The Add Nofaults command places a no-fault setting on either a single specified pin or on all pins of a specified instance or module. If the pathname is a pin, then DFTAdvisor ignores the fault on only that pin. If the pathname is an instance, then the tool ignores all pin faults on the top-level of that instance, along with all the pin faults underneath that instance (if it is a hierarchical instance). If the pathname is a module, then the tool ignores all pin faults on the top-level of the module, along with all the pin faults on all instances and pins underneath that module for every occurrence of that module in the design. The nofaults that you create with the Add Nofaults command only exist for the current DFTAdvisor session.

DFTAdvisor recognizes the no-fault setting on pins and instances through two different tagging processes. The first process involves interactively using the Add Nofaults command (which you can use on pins and instances); the second process involves using the no-fault DFT library attribute (which you can only use on pins, not on instances).

Arguments

- *pathname*
  A required string that specifies either the pin pathname or the instance pathname for which you want to assign no-fault settings. If the pathname you specify is an instance or module pathname, you must use the -Instance or -Module switch.

- -Instance | -Module
  An optional switch specifying that the *pathname* argument is an instance pathname or a module pathname. If the pathname is a module, all instances and
pins within the module are affected. If neither switch is specified, the default pathname is a pin pathname.

- **-Stuck_at 01 | 0 | 1**

An optional switch and literal pair that specifies to which stuck-at values you want to assign a nofault setting. The choices for stuck-at values are as follows:

- **01** — A literal that specifies the placement of a nofault setting on both the “stuck-at-0” and “stuck-at-1” faults. This is the default.

- **0** — A literal that specifies the placement of a nofault setting only on the “stuck-at-0” faults.

- **1** — A literal that specifies the placement of a nofault setting on the “stuck-at-1” faults.

- **-Keep_boundary**

An optional switch that specifies that nofaults are applied to the pins inside of the specified instance or module, but faults are still allowed at the boundary pins of the specified instances or modules. This option does not apply to nofaults on pin pathnames.

### Examples

The following example first tags all the pin faults on and below an instance. The example then tags the fault on a specific pin.

```
add clocks 0 clock
add nofaults i_1006 -instance
add nofaults i_1_16/df0/q
set system mode dft
run
```

### Related Commands

- [Delete Nofaults](#)
- [Report Nofaults](#)
Add Nonscan Instances

Scope: All modes, except only Dft mode for the -Control_signal option.

Usage

ADD NONscan Instances \textit{pathname}... [-INStance | -Control_signal | -Module]

Description

Specifies for DFTAdvisor to ignore the specified instances, all instances controlled by the specified control pin, or all instances within the specified module, when identifying and inserting the required scan elements and test logic.

The Add Nonscan Instances command causes DFTAdvisor to ignore the specified instances, instances controlled by the specified control signals, or all instances within the specified module when you execute both the scan identification process with the Run command and the scan insertion process with the Insert Test Logic command.

Also, DFTAdvisor will not perform the scannability rule checks on nonscan instances unless you execute the Set Nonscan Handling command.

If you wish to flag all instances of a certain library model as nonscan, you can use the Add Nonscan Models command as a short cut.

When adding nonscan instances with control signals, DFTAdvisor adds all scannable instances controlled by the control signals to the nonscan instance list. Control pins can be any clock, set, or reset signal (primary input) that controls the contents of sequential instances. You can use the Report Control Signals command to see the pins that control the sequential instances in the design.

If TIE0 and TIE1 nonscan cells are scannable, they are considered for scan. However, if these cells are used to hold off sets and resets of other cells so that another cell can be scannable, you must use this command to make them nonscan.

Arguments

- \textit{pathname}
  
  A required repeatable string that specifies the pathnames of the sequential instance or control signals (that control instances) which you want DFTAdvisor to ignore. If the instance is hierarchical, then all instances beneath it are also flagged as nonscan.
- **INStance** | -Control_signal | -Module

A switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, ...)” where clkgen is the module name. You can only use the -Control_signal option in Dft mode. The default is instances.

### Examples

The following example specifies that DFTAdvisor ignore the sequential i_1006 instance when identifying and inserting the required scan circuitry:

```plaintext
add nonscan instances i_1006
```

### Related Commands

- **Delete Nonscan Instances**
- **Insert Scan Chains**
- **Insert Test Logic**

<table>
<thead>
<tr>
<th>Command</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Nonscan Instances</td>
<td>Report Nonscan Instances</td>
</tr>
<tr>
<td>Insert Scan Chains</td>
<td>Set Nonscan Handling</td>
</tr>
<tr>
<td>Insert Test Logic</td>
<td></td>
</tr>
</tbody>
</table>
Add Nonscan Models

Scope: All modes

Usage

ADD NONscan Models *model_name*...

Description

Instructs DFTAdvisor to ignore all instances of the specified sequential DFT library model when identifying and inserting the required scan elements and test logic into the design.

The Add Nonscan Models command causes DFTAdvisor to ignore the instances of the specified models when you execute both the scan identification process with the Run command and the scan insertion process with either the Insert Scan Chains command or the Insert Test Logic command.

Also, DFTAdvisor will not check the scannability on the instances instantiated from the added nonscan models unless you specify checking using the Set Nonscan Handling command.

If you wish to flag individual instances as nonscan, you can use the Add Nonscan Instances command.

Arguments

- *model_name*

  A required repeatable string that specifies the model names that you want DFTAdvisor to ignore. You must enter the model names as they appear in the DFT library.

Examples

The following example specifies that DFTAdvisor should not substitute any of the instances of the DFT library model named *d_flip_flop* with the equivalent scan replacement DFT library model.

  add nonscan models *d_flip_flop*
Related Commands

- Delete Nonscan Models
- Insert Scan Chains
- Insert Test Logic
- Run
- Report Nonscan Models
- Set Nonscan Handling
Add Notest Points

Scope: DFT mode

Usage

ADD NOtest Points {pin_pathname... | instance_pathname...} | -Path filename

Description

Adds circuit points to list for exclusion from testability insertion.

The Add Notest Points command excludes the specified cell output pins, excludes all output pins on the specified instance, or excludes delay paths from use as controllability and observability insertion points. If the selected pin is already a control or observe point, an error occurs when you issue this command.

If you specify a path file that contains delay paths, each gate contained in a path is marked so that no test point can be added to the output of that gate. This prevents control and observation points from being added to a critical path and thus prevents increasing the load on any of these gates in that path. The format of the file is the same as file loaded into FastScan with the Load Paths command. For more information on the format, refer to “The Path Definition File” in the Scan and ATPG Process Guide.

You can use the Report Notest Points command to display all the pins in this list or list the defined critical paths.

You use this command primarily for implementing Built-In Self-Test (BIST) circuitry.

Arguments

- pin_pathname
  A required repeatable string that lists the output pins that you do not want to use for controllability and observability insertion.

- instance_pathname
  A required repeatable string that lists the instances whose output pins you do not want to use for controllability and observability insertion. All output pins within that (hierarchical) instance are added to the list of pins that should be excluded from consideration.
• **-Path filename**
  A required switch and filename pair that specifies the pathname to a file that contains critical path information. For more information on the format of the file, refer to “The Path Definition File” in the *Scan and ATPG Process Guide*.

**Examples**

The following example first sets up the test point identification parameters, then specifies output pins *tr_io* and *ts_i* that DFTAdvisor cannot use for testability insertion:

```
set control threshold 4
set observe threshold 4
setup test_point identification -control 9 -obs 20 -patterns 32000 -cshare 16 -oshare 16 -base simulation
set system mode dft
setup scan identification none
add notest points tr_io ts_i
run
```

**Related Commands**

- Delete Notest Points
- Report Notest Points
- Setup Test_point Identification
Add Output Masks

Scope: Setup mode

Usage
ADD Output Masks primary_output... [-Nohold | -Hold {0 | 1}]

Description
Instructs DFTAdvisor to mask, and optionally maintain a constant logic level on, the specified primary output pins during the scan identification analysis.

DFTAdvisor uses primary output pins as the observe points during the scan identification process. When you mask a primary output pin, you inform DFTAdvisor to mark that pin as an invalid observation point during the identification process.

There are specific design practices that mask specific primary output pins in order to group the sequential elements associated with those primary outputs. For example, if you are using partition scan. For more information on partition scan refer to “Understanding Partition Scan” in the Scan and ATPG Process Guide.

You can set a default mask for all output and bi-directional pins using the Setup Output Masks command. You can add a hold value to a default mask with the Add Output Masks command, or remove a hold value using the Delete Output Masks command. To turn off the default masks for all output pins, you must use the Setup Output Masks command with the Off literal.

Arguments
- primary_output
  A required repeatable string that specifies the names of the primary output pins you want to mask. When you enter the primary_output argument without the -Hold switch, DFTAdvisor marks the specified pins as invalid observation points during the identification process.

  If you specify the -Hold switch, then, in addition to masking the specified pins, DFTAdvisor also applies and maintains the specified (-Hold) state value on the pins.
• -Nohold
   An optional switch specifying that you want DFTAdvisor to allow any state value on the primary output pin. This is the default.

• -Hold 0 | 1
   An optional switch and literal pair that specify the particular state that you want DFTAdvisor to maintain on the primary output pin. If you use this switch, you can explicitly specify that the output pin is driving the inactive value. The choices for the hold literal value are as follows:
   
   0 — A literal that specifies to maintain a low logic state on the primary output pin.
   
   1 — A literal that specifies to maintain a high logic state on the primary output pin.

Examples

The following example first sets up DFTAdvisor to recognize only the partition cells during the scan identification run. The invocation default identification type is sequential scan. Next, the example specifies the primary output pins that the Add Output Masks command associates with the partition scan cells.

   setup scan identification partition_scan
   add output masks out1 out2 out3

When you issue the Run command later in the session, DFTAdvisor identifies all the sequential elements that are observable only through the primary output pin that you masked. Then, when you issue the Insert Test Logic command, DFTAdvisor stitches all those scan cells it previously identified as being in the partition into one partition scan chain.

Related Commands

- Analyze Output Observe
- Delete Output Masks
- Report Output Masks
- Setup Output Masks
- Setup Scan Identification
Add Pin Constraints

Scope: Setup mode

Usage

ADD PIn Constraints primary_input_pin \{ C0 | C1 | CZ | CX | CR0 | CR1 \}

Description

Specifies that DFTAdvisor hold the input pin at a constant state during the rules checking and loop cutting processes.

DFTAdvisor performs the rules checking and loop cutting processes when you issue the Set System Mode Dft command. During the rules checking process there are times that you want to hold a pin at a constant state. For example, you may want to hold a test enable pin at the active state to keep the chip in test mode.

Note

This command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

If you are using the partition scan type, DFTAdvisor also uses pin constraints that are set to unknown as a way to flag partition primary inputs that are uncontrollable from the higher chip-level primary inputs. When DFTAdvisor performs the scan identification process during the Run command, it adds to the scan candidate list the sequential cells that are controllable through the primary input pin; those which you constrained to an unknown value. For more information on partition scan, refer to “Understanding Partition Scan” in the Scan and ATPG Process Guide.

You can set a default pin constraint value for all input and bi-directional pins using the Setup Pin Constraints command. The pin constrains set by the Setup Pin Constraints command can be overridden by the values set with the Add Pin Constraints command. You can remove an override of a default pin constraint using the Delete Pin Constraints command. To remove the default pin constraint for all input pins, you should use the Setup Pin Constraints command with the None literal.
Arguments

- **primary_input**
  
  A required string that specifies the primary input pin that you want DFTAdvisor to hold at the constant_value.

- **C0 | C1 | CZ | CX | CR0 | CR1**
  
  A literal specifying application of the constant value with which you want to constrain the primary_input pin. The constraint choices are as follows:

  - **C0** — A literal that specifies application of the constant 0 to the chosen primary input pins.
  - **C1** — A literal that specifies application of the constant 1 to the chosen primary input pins.
  - **CZ** — A literal that specifies application of the constant Z (high-impedance) to the chosen primary input pins.
  - **CX** — A literal that specifies application of the constant X (unknown) to the chosen primary input pins.
  - **CR0** — A literal that specifies a constant that returns to 0; DFTAdvisor uses this constant only when formatting the patterns. The ATPG process treats CR0 as a C0.
  - **CR1** — A literal that specifies a constant that returns to 1; DFTAdvisor uses this constant only when formatting the patterns. The ATPG process treats CR1 as a C1.

Examples

The following example illustrates how to hold two primary input pins to constant values:

```
add pin constraints kgmt c1
add pin constraints dsint c0
```

Related Commands

- Add Seq_transparent Constraints
- Report Pin Constraints
- Analyze Input Control
- Setup Pin Constraints
- Delete Pin Constraints
Add Pin Equivalences

Scope: Setup mode

Usage

ADD PIN Equivalences primary_input_pin... [-Invert] primary_input_pin_ref

Description

Specifies to hold the specified primary input pins at a state either equal to or inverted in relationship to the state of another primary input pin during the rules checking.

The rules checking occurs with the Set System Mode Dft command. During these rule checks there are times that a pin needs to be held at the same (or opposite) state in reference to another pin. For example, using a pin equivalence can be helpful when setting up the appropriate off-states for all clocks within the design.

Note

This command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

Arguments

- **primary_input_pin**
  
  A required repeatable string specifying a list of primary input pins whose values you want to either equal or invert with respect to primary_input_pin_ref.

- **-Invert**
  
  An optional switch that specifies for DFTAdvisor to hold the primary_input_pin value to the opposite state of the primary_input_pin_ref value. If you use this switch, you must enter it immediately prior to the primary_input_pin_ref value.

- **primary_input_pin_ref**
  
  A required string specifying the name of the primary input pin whose value you want DFTAdvisor to use when determining the state value of
primary_input_pin. You can immediately precede this string with the -Invert switch to cause DFTAdvisor to hold the primary_input_pin value to the opposite state of the primary_input_pin_ref value.

**Examples**

The following example restricts the first primary input (indata2) to have an inverted value with respect to the second primary input (indata4):

    add pin equivalences indata2 -Invert indata4

**Related Commands**

Delete Pin Equivalences    Report Pin Equivalences
Add Primary Inputs

Scope: Setup mode

Usage

ADD PRimary Inputs *net_pathname* ... [-Cut] [-Module]

Description

Adds a primary input to the net.

The Add Primary Inputs command adds an additional primary input to each specified net path. Once added, the tool designates them as user class primary inputs, as opposed to the primary inputs described in the original netlist, which it designates as system class primary inputs. Use the -Cut option to disconnect the original drivers of the net so that the added primary input becomes the only driver of the net. Otherwise, if there are other drivers besides the newly added primary input, the tool treats this net as a wired net. You can display the user class, system class, or full classes of primary inputs using the Report Primary Inputs command.

Arguments

- *net_pathname*
  A required repeatable string specifying the pathname of the pins to which you want to add primary inputs.

- -Cut
  An optional switch that specifies disconnection of the original drivers of the net, making the added primary input the only driver of the net. The design must be flattened with the Flatten Model command prior to using this option.

- -Module
  An optional switch that specifies addition of the primary input to the specified nets in all modules. Only one primary input is added to the design for that net when many occurrences of that net occurs in the modules.
Examples

The following example adds two new primary inputs to the circuit and places it in the user class of primary inputs:

```
add primary inputs indata2 indata4
```

Related Commands

- Delete Primary Inputs
- Report Primary Inputs
Add Primary Outputs

Scope: Setup mode

Usage

ADD PRimary Outputs net_pathname...

Description

Adds a primary output to the net.

The Add Primary Outputs command adds an additional primary output to each specified net. Once added, the tool defines them as user class primary outputs. The tool defines the primary outputs described in the original netlist as system class primary outputs. You can display the user class, system class, or full classes of primary outputs using the `Report Primary Outputs` command.

Arguments

- net_pathname
  
  A required repeatable string that specifies the nets to which you want to add primary outputs.

Examples

The following example adds a new primary output to the circuit and places it in the user class of primary outputs:

`add primary outputs outdata1`

Related Commands

- Delete Primary Outputs
- Report Primary Outputs
Add Read Controls

Scope: Setup mode

Usage
ADD REad Controls \{0 | 1\} \textit{primary_input_pin}...

Description
Adds an off-state value to specified RAM read control lines.
The Add Read Controls command defines the circuit read control lines and assigns their off-state values. The off-state value of the pins that you specify must be sufficient to keep the RAM outputs stable. You cannot use clocks, constrained pins, or equivalent pins as read control lines.

Arguments
- 0
  A literal specifying 0 as the off-state value for the RAM read control lines.
- 1
  A literal specifying 1 as the off-state value for the RAM read control lines.
- \textit{primary_input_pin}
  A required repeatable string that lists the primary input pins you want to designate as RAM read control lines and to which you are assigning the given off-state value.

Examples
The following example assigns an off-state value of 0 to two read control lines, r1 and r2:

```
add clocks 0 clk
add read controls 0 r1 r2
set system mode dft
run
```

Related Commands
Delete Read Controls Report Read Controls
Add Scan Chains

Scope: Setup mode

Prerequisites: You must define the scan chain group with the Add Scan Groups command prior to using this command. If multiple scan chains are in the same scan group, you must load the chains in parallel.

Usage

ADD Scan Chains chain_name group_name primary_input_pin
primary_output_pin

Description

Specifies a name for a preexisting scan chain within the design.

The Add Scan Chains command defines a scan chain that exists in the design. A scan chain references the name of a scan chain group, which you must define prior to issuing this command.

A preexisting scan chain is a serially connected set of scan cells that have been stitched together in a previous scan insertion operation or that you created manually. In order for DFTAdvisor to be able to correctly insert scan elements in the remainder of the design or to run rules checking on existing scan circuitry, you need to notify DFTAdvisor of this circuitry so it can treat those elements in the chain differently than the elements in the rest of the design.

When you use this command to notify DFTAdvisor of a preexisting scan chain, DFTAdvisor removes the scan cells in the scan chain from the eligible scan elements list. Because the tool removes the scan cells in the scan chain from the scan candidate list, the rules checker does not perform the usual scannability checks on those scan cells. However, the rules checker does perform additional rules checking based on the declaration of preexisting scan elements that pertain to the checking of the validity and the operation of that scan chain.

If the design does have a preexisting scan chain that you declare with the Add Scan Chains command, you need to provide information on the operation of that scan chain in a test procedure file.
Arguments

- **chain_name**
  A required string specifying the name of the preexisting scan chain you want added to the scan group.

- **group_name**
  A required string specifying the name of the scan chain group to which you are adding the scan chain.

- **primary_input_pin**
  A required string specifying the primary input pin of the scan chain.

- **primary_output_pin**
  A required string specifying the primary output pin of the scan chain.

Examples

The following example defines two scan chains (chain1 and chain2) that belong to the same scan group (group1):

```
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 testout2
add scan chains chain2 group1 indata4 testout4
```

Related Commands

- Add Scan Groups
- Delete Scan Chains
- Report Scan Chains
- Ripup Scan Chains
Add Scan Groups

Scope: Setup mode

Usage

ADD SCan Groups group_name test_procedure_filename

Description

Adds one scan chain group to the system.

The Add Scan Groups command has two different uses. The main use of this command is to specify a group name for a set of preexisting scan chains within the design. You can only add one scan chain group within a DFTAdvisor session. Additionally, you can use this command in DFTAdvisor to provide necessary initialization values to non-scan memory elements.

If the design has preexisting scan chains, those scan chains must belong to a scan group. The Add Scan Group command also specifies the corresponding test procedure file for that scan group. A sequence of procedures in the test procedure file defines the operation of the scan chains within a scan group. The scan group must have a corresponding test procedure file.

If the design does not have preexisting scan chains, but you need to initialize some non-scan memory elements specifically for scannability checking, you can specify “dummy” as the group_name along with a test_procedure_filename. Then, within the test procedure file, you can use the optional test_setup procedure to initialize the non-scan memory elements. As a result, DFTAdvisor will model the non-scan memory elements as a TIE1 or TIE0.

You must also define these same elements as non-scan using the Add Nonscan Instance command.

Arguments

- group_name
  A required string that specifies the name of the scan chain group that you want to add to the system.
• **test_procedure_filename**
  
  A required string that specifies the name of the test procedure file that contains the information for controlling the scan chains in the specified scan chain group.

**Examples**

The following example defines a scan chain group, group1, which loads and unloads a set of scan chains, chain1 and chain2, by using the procedures in the file, scanfile:

```plaintext
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 testout2
add scan chains chain2 group1 indata4 testout4
```

**Related Commands**

- Add Scan Chains
- Delete Scan Groups
- Report Scan Groups
Add Scan Instances

Scope: All modes

Usage

ADD SCAn Instances *pathname* ... [-INStance | -Control_signal | -Module]
[-INPut | -Output | {-Hold {0 | 1}}]

Description

Specifies that DFTAdvisor add the specified instance, all instances controlled by
the specified control pin, or all instances within the specified module, to the
scannable instance list.

If DFTAdvisor is only inserting partial scan, the Add Scan Instances command
allows a way of ensuring that DFTAdvisor includes specific sequential instances
in the scan list. DFTAdvisor generates the scan list during the scan identification
process (which you execute with the Run command in the Dft mode). However, if
the instance you specify does not pass the rules checking process, DFTAdvisor
cannot include it in the identified scan list.

If you do not issue the Add Scan Instances command with partial scan,
DFTAdvisor chooses the sequential instances it includes in the identified scan list
based on the settings you specified with the Setup Scan Identification command
(or uses the default argument values for that command).

If DFTAdvisor is in the Dft mode, the Add Scan Instances command can accept
scannable instance pathnames, control signal pins, or module names as arguments.
Scannable instances are those sequential instances that passed all the scannability
checks run by the Design Rule Checker when you entered the Dft mode.

When adding scan instances with control signals, DFTAdvisor adds all scannable
instances controlled by the control signals to the identified scannable instance list.
Control pins can be any clock, set, or reset signal (primary input or output) that
controls the contents of sequential instances. You can use the Report Control
Signals command to see the pins that control the sequential instances in the
design.
Command Dictionary

Add Scan Instances

Arguments

• **pathname**
  A required repeatable string that specifies the pathnames of the sequential instances or control signals (that control instances) which you want to add to the scan instance list. If the instance is hierarchical, then the tool also adds all sequential instances beneath it to the scan list.

• **-INStance | -Control_signal | -Module**
  A switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, ...)” where clkgen is the module name. You can only use the -Control_signal option in Dft mode. The default is -Instance.

• **-INPut | -Output | {-Hold {0 | 1}}**
  An optional switch that the scan instances are added as input or output partition scan cells. If you specify the -Hold option, then you must also supply a high (1) or low (0) literal to define hold 0 or hold 1 output partition scan cells. If none of these options are specified, the added scan instances are considered as regular scan cells.

Examples

The following example first specifies two sequential instances that the user wants to ensure are in the identified scan list (assuming they pass rules checking). In the remainder of the example, you enter Dft mode, specify that DFTAdvisor is to choose the 50 percent of the eligible scan elements that maximizes the fault coverage, and then run the scan identification process.

```plaintext
add scan instances i_1006 i_1007
set system mode dft
setup scan identification sequential atpg -percent 50
run
```

Because of the previous setup in this example, when DFTAdvisor runs the scan identification process it chooses the optimal fifty percent of eligible scan instances, but always includes i_1006 and i_1007 within that fifty percent.
Related Commands

<table>
<thead>
<tr>
<th>Delete Scan Instances</th>
<th>Setup Scan Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Scan Instances</td>
<td></td>
</tr>
</tbody>
</table>
Add Scan Models

Scope: All modes

Usage

ADD SCAN Models model_name...

Description

Specifies that DFTAdvisor is to flag every instance of the named DFT library model for inclusion into the identified scan list.

If DFTAdvisor is only inserting partial scan, the Add Scan Models command allows a way of ensuring that DFTAdvisor includes all instances of a specific sequential DFT library model in the scan list. DFTAdvisor generates the scan list during the scan identification process (which you initiate with the Run command in the Dft mode). DFTAdvisor then replaces all instances in the scan list with the equivalent DFT library scan model during scan synthesis (which you initiate with the Insert Test Logic command within the Dft mode).

If you do not issue the Add Scan Models command with partial scan, DFTAdvisor chooses the sequential instances it includes in the scan list based on the settings you specified with the Setup Scan Identification command (or uses the default argument values for that command).

If DFTAdvisor is in the Dft mode, the Add Scan Models command only flags the scannable instances of the specified library model for inclusion into the scan list. Scannable instances are those sequential instances that passed all the scannability checks run by the Design Rule Checker when you entered the Dft mode. So, if there are instances of the specified library model that were not put into the eligible scan list because they failed some of the design rules checks, DFTAdvisor would not include those instances in the identified scan list.

The Add Scan Models command flags all instances of the specified sequential DFT library model, where the Add Scan Instances Command only individually flags sequential design instance(s).
Add Scan Models

Arguments

- **model_name**
  A required repeatable string that specifies the model names that you want to add to the scan model list. Enter the model names as they appear in the DFT library.

Examples

The following example flags all instances of the specified DFT library model for inclusion into the scan list.

```plaintext
add scan models dff1a
set system mode dft
setup scan identification sequential atpg -percent 50
run
```

Because of the previous setup in this example, when DFTAdvisor runs the scan identification process it chooses the optimal fifty percent of eligible scan instances, ensuring that it includes all eligible instances of the dff1a model in that fifty percent of identified scan instances.

Related Commands

- Delete Scan Models
- Report Scan Models
Add Scan Pins

Scope: All modes

Usage

ADD SCan Pins chain_name scan_input_pin scan_output_pin [-Clock pin_name] [-Cut] [-Registered]

Description

Declares the name of a scan chain at the top-level module and assigns the corresponding scan input pin, scan output pin, and optionally, the scan clock pin that you wish to associate with that chain.

If you specify signal names for the scan input or scan output pins that do not currently reside in the design, DFTAdvisor generates a warning and then adds the appropriate pins when you perform the scan synthesis process with the Insert Test Logic command.

If you do not issue either this command, or the Setup Scan Pins command, DFTAdvisor uses the default names when generating the scan chain. The default name of the scan chain is chain1, with the corresponding scan input pin named scan_in1, and the corresponding scan output pin named scan_out1. If DFTAdvisor inserts multiple scan chains, the default naming increments the suffix number of each argument by one for each chain (such as chain2, scan_in2, and scan_out2).

You can optionally use the Setup Scan pins command to globally set the default naming conventions for scan chains that do not use existing design pins. The Add Scan Pins command just specifies the naming for a specific scan chain.

Arguments

- chain_name

A required string that specifies the name of the scan chain with which you want DFTAdvisor to associate the scan_input_pin and scan_output_pin names.
Add Scan Pins

**Command Dictionary**

- **scan_input_pin**

  A required string that specifies the scan input pin name of the scan chain. This pin can be a top-level input pin, top-level bidirectional pin, or an internal signal.

  In addition to a primary input pin name (for instance, scan_in), you can also specify an internal instance pin path name (for example, /I116/q). However, if you specify an internal instance pin path name as the `scan_input_pin` value, that pin path name must be an output pin of an instance. If you specify an internal pin path name and DFTAdvisor cannot trace through a simple path (only through inverters or buffers) back to a primary input of the design, DFTAdvisor cannot generate the test procedure file and the dofile with the Write Atpg Setup command. Therefore, you must manually develop these two files before running ATPG with either FastScan or FlexTest.

  If the pin is a top-level bidirectional pin, DFTAdvisor assumes that you configured the pin to be in input mode during the scan test and does not check for correct configuration.

  If -Registered is specified, the `scan_input_pin` is the output of the DFF head register.

- **scan_output_pin**

  A required string that specifies the scan output pin name of the scan chain. This pin can be a top-level output pin, top-level bidirectional pin (single driver), or an internal signal. If the scan-out pin you specify is driven by multiple instances, DFTAdvisor adds a mux gate to multiplex the original functional driver with the driver from the last scan cell of the scan chain.

  In addition to a primary output pin name (such as scan_out), you can also specify an internal instance pin path name (such as /I116/d). However, if you specify an internal instance pin path name as the `scan_output_pin` value, that pin path name must be an input pin of an instance. If you specify an internal pin path name and DFTAdvisor cannot trace through a simple path (only through inverters or buffers) forward to a primary output of the design, DFTAdvisor cannot generate the test procedure file and the dofile with the Write Atpg Setup command. Therefore, you must manually develop these two files before running ATPG with either FastScan or FlexTest.
If the pin is a top-level bidirectional pin, DFTAdvisor assumes that you configured the pin to be in output mode during the scan test and does not check for correct configuration.

If -Registered is specified, the scan_output_pin is the input of the DFF tail register.

• -Clock *pin_name*
  
  An optional switch and string pair that specifies the pin name of the clock that you want DFTAdvisor to assign to the scan chain. You must have predefined this pin as a scan clock by using the Add Clocks command.

• -Cut

  An optional switch that specifies to remove the existing driver attached to the specified scan out and the last scan cell is then connected to the scan out.

• -Registered

  An optional switch that specifies to attach head and tail DFF registers to the scan chain. For additional information, refer to “Attaching Head and Tail Registers to the Scan Chain” in the Scan and ATPG Process Guide.

**Examples**

The following example assigns a scan input and scan output name to the scan chain:

```plaintext
add clocks 0 clk
set system mode dft
// run scan identification
run
add scan pins chain1 si so -clock clk
// scan insertion
insert test logic
```

**Related Commands**

- Delete Scan Pins
- Report Scan Pins
- Setup Scan Pins
Add Seq_transparent Constraints

Scope: Setup mode

Usage

ADD SEq_transparent Constraints {C0 | C1} model_name pin_name...

Description

Specifies the enable value of a clock enable that internally gates the clock input of a non-scan cell for sequential transparent scan identification.

Designs sometimes contain circuitry where the clock enable signal internally gates the clock input of a non-scan cell. In these cases, the clock enable must hold at a certain value in order for the cells to behave as sequentially transparent. The clock enable input of a cell may come from primary inputs or other memory elements.

If the sources of a clock enable input come from the output of other memory elements, DFTAdvisor identifies those memory elements for scan during the scan_sequential identification run.

Arguments

• C0 | C1
  A required literal that specifies the application of the constant 0 or 1 to the pin_name.

• model_name
  A required string that specifies the DFT library model to whose pin_name you want to apply the specified C0 or C1 constant.

• pin_name
  A required repeatable string that specifies the clock enable pin name for the model_name specified and that to which DFTAdvisor will apply the specified constant.

Examples

Assume that Figure 2-2 shows the DFT library model used in your design and you want DFTAdvisor to consider its instantiation as a sequential transparent cell. The
following example defines the sequential transparent constraint for EN and then verifies whether EN can be held at C1 during the sequential transparent identification run:

```
add seq_transparent constraints c1 gdff en
set system mode dft
setup scan identification seq_transparent
run
```

**Figure 2-2. DFT Library Model GDFF**

**Related Commands**

- Add Pin Constraints
- Delete Seq_transparent Constraints
- Report Pin Constraints
Add Sub Chains

Scope: Setup mode

Usage

ADD SUb Chains object_name subchain_name scan_input_pin
scan_output_pin length scan_type [-Module | -Instance]
[-TEN test_enable_pin {0 | 1}] [-TCLK test_clock_pin {0 | 1}]
[-SET set_pin {0 | 1}] [-RESET reset_pin {0 | 1}]

Description

Specifies the name of a preexisting scan chain that exists entirely within a module or instance within a hierarchical design.

If you create the design’s scan on a block-by-block basis, then each instance or module has its own preexisting scan chain(s). At the higher design level, DFTAdvisor does not automatically consider the chains that it inserts at the lower level as scan sub-chains. The Add Sub Chains command allows you to incorporate those scan sub-chains into the top-level scan chain(s) during the stitching process. However, DFTAdvisor does not run the rules checker on the preexisting scan sub-chain(s).

If you perform block-by-block test synthesis at lower design levels, DFTAdvisor can write the sub-chain setup information to a file with the Write Subchain Setup command. You can then read in this setup file that contains the Add Sub Chains command as part of the setup and avoid having to manually define the preexisting scan sub-chains at the higher design level.

You can either leave sub-chain input pins floating or tie them to 0 or 1. If you tie a sub-chain input pin, DFTAdvisor removes the tie and connects it to the previous scan out of the scan cell.

Arguments

- **object_name**
  
  A required string that specifies either the pathname of an instance or the name of a module. If you provide the pathname of an instance, you must also use the -Instance switch.
• **subchain_name**
  A required string that specifies the name that you want to assign a preexisting scan sub-chain.

• **scan_input_pin**
  A required string that specifies the scan input pin of the preexisting scan sub-chain.

• **scan_output_pin**
  A required string that specifies the scan output pin of the preexisting scan sub-chain.

• **length**
  A required integer that specifies the number of scan cells in the preexisting scan sub-chain.

• **scan_type**
  A required multiple entry argument that specifies the scan type and control of the preexisting scan sub-chain. The valid types of scan values are as follows:

    **Mux_scan scan_enable** [ {-Clock pin_name} | {-Subclock pin_name} ] —
    A literal and string pair that specifies mux-DFF style of scan cells and the name of the scan enable pin for the preexisting scan sub-chain. You can also specify the clock domain for the sub-chains by using one of the following switches:

    - **-Clock pin_name** — An optional switch and string pair specifying the clock pin name on the top module which clocks the defined sub chain.

    - **-Subclock pin_name** — An optional switch and string pair specifying the clock pin name on the sub module where the defined sub chain resides.

    If you do not specify either of these options, DFTAdvisor considers the clock domain of the sub chain as undefined.

    **Clocked_scan scan_clock** — A literal and string pair that specifies clocked-scan style of scan cells and the name of the scan clock for the preexisting scan sub-chain.
**Lssd master_clock slave_clock** — A literal and two string triplet that specifies Level-Sensitive Scan Design and the names of the master and slave clocks, respectively, for the preexisting scan sub-chain.

- **-Module | -Instance**
  An optional switch that specifies for DFTAdvisor to interpret the value of the `object_name` argument as a module name (-Module) or as a pathname to an instance (-Instance). The default is -Module.

- **-TEN test_enable_pin \{0 | 1\}**
  An optional switch, string and literal pair that specifies the test enable pin added in the sub-module which is to be connected to the corresponding pin in the top module. The active value can be 0 or 1.

- **-TCLK test_clock_pin \{0 | 1\}**
  An optional switch, string and literal pair that specifies the test clock pin added in the sub-module which is to be connected to the corresponding pin in the top module. The off-state value can be 0 or 1.

- **-SET set_pin \{0 | 1\}**
  An optional switch, string and literal pair that specifies the set pin added in the sub-module which is to be connected to the corresponding pin in the top module. The offstate value can be 0 or 1.

- **-RESET reset_pin \{0 | 1\}**
  An optional switch, string and literal pair that specifies the reset pin added in the sub-module which is to be connected to the corresponding pin in the top module. The offstate value can be 0 or 1.

**Examples**

The following example defines a preexisting scan sub-chain:

```
add sub chain addr subc1 /scan_in1 /scan_out1 8 mux_scan /scan_en
-module
report sub chains
mux_scan: addr subc1 scan_in1 scan_out1 scan_en
```
Related Commands

Delete Sub Chains  Report Sub Chains
Insert Test Logic  Write Subchain Setup
Add Test Points

Scope: Setup and DFT mode

Prerequisites: You must define the dftlib_model with either the Add Cell Models command or the cell_type attribute within the DFT library.

Usage

ADD TEst Points tp_pin_pathname [{Control model_name input_pin_pathname [mux_sel_input_pin] [scan_cell]} | {Observe output_pin_pathname [scan_cell]} | {Lockup lockup_latch_model clock_pin [-INVert | -NOInvert]}]

Description

Specifies explicitly where DFTAdvisor is to place a user-defined test point to improve the design’s testability either through better controllability or observability.

The Add Test Points command allows you to manually specify the location of a test point and whether that point is for control or observation. This command also allows you to specify whether to insert a scan cell for control or observe purposes at the test point location in addition to the control logic. An additional use of this command is for manually specifying the location of lockup latches to control timing problems on merged scan chains.

The Add Test Points command operates independently of the Set Test Logic command. Set Test Logic causes DFTAdvisor to automatically insert test logic during the scan synthesis process, based upon the analysis DFTAdvisor performs during the rules checking process.

The Add Test Points command also works independently of the automatically system-defined test points. For more information on using the automatic test point functionality, refer to “Understanding Test Points” in the Scan and ATPG Process Guide.

You can manually create user-defined test points with the Add Test Points command to increase the testability of the design. For example, a test point can be useful in disabling latch loops or cutting non-duplicating combinational loops. Note that this command only specifies the location at which to add the test point.
If you specify the `scan_cell` option, DFTAdvisor adds a scan cell for either control or observe purposes (depending on which you selected). DFTAdvisor inserts an instance of the `scan_model` at the same time it inserts the rest of the scan elements, which is when you issue the **Insert Test Logic** command.

**Arguments**

- **tp_pin_pathname**
  A required string that specifies the location where you want DFTAdvisor to insert the control or observe test point.

- **Control model_name input_pin_pathname [mux_sel_input_pin] [scan_cell]**
  The Control test point argument specifies that the test point is for control purposes.

  - **model_name** — A string that specifies the DFT library model that you want DFTAdvisor to place an instance of at the location specified by `tp_pin_pathname`. Before you can use the Add Test Points command, you must use either the **Add Cell Models** command or the `cell_type` DFT library attribute to define the DFT library model that corresponds to the model type you want DFTAdvisor to insert. The valid cell model types include AND, OR, INV, BUF, NAND, NOR, XOR and MUX.

  - **input_pin_pathname** — A string that specifies the pathname of the pin to which you want to connect the other input of the gate specified by the `model_name` argument. The pathname can be either to an existing primary input pin, an internal driver pin, or a currently nonexistent pin. If the pin does not currently exist in the design, DFTAdvisor transcripts a message when you issue this command, and then creates a new primary input pin with the specified name during the insertion of the scan chain(s).

If you use the `-INTscan scan_model` option of the `scan_cell` argument to insert a scan cell at the test point location, DFTAdvisor uses the `input_pin_pathname` you specify as the input pin for the scan cell’s clock, as shown in Figure 2-3.

  - **mux_sel_input_pin** — An optional string that is needed only when the `model_name` argument type is a MUX. It is not needed for inserting a
buffer or inverter. This argument specifies where DFTAdvisor is to connect the selector input of the multiplexer.

---

**Figure 2-3. Control Example**

- **scan_cell** — An optional switch that specifies whether DFTAdvisor places a scan cell at the control test point. Note that these are new scan cells, not scan replacements for existing sequential elements. They are connected into chain(s) during insertion of test logic. If the design contains no scan, the test point scan cells are connected into one or more scan chains, depending on their clock pins. The following switches are the valid options:

  - **-NONE** — A switch that specifies not to insert a scan cell at the test point. This is the default.

  - **-INT scan scan_model** — A switch and string pair that specifies the DFT library SCANCELL type model that you want DFTAdvisor to insert at the test point. If you use this option, you must first define the `scan_model` with the Add Cell Models command or the `cell_type` DFT library attribute. If the test point is for control, DFTAdvisor uses the `input_pin_pathname` you specify as the input pin for the scan cell’s clock.
• **Observe output_pin_pathname** [scan_cell]

The Observe test point argument specifies for DFTAdvisor to place an observe point at the location specified by the value of the tp_pin_pathname argument.

- **output_pin_pathname** — A string that specifies the pathname of the primary output pin that you want DFTAdvisor to connect to the observe point. If the primary output pin does not currently exist in the design, DFTAdvisor creates a new primary output pin with the specified name during the insertion of the scan chain(s).

If you use the -INTscan scan_model option of the scan_cell argument to insert a scan cell at the test point location, DFTAdvisor uses the output_pin_pathname you specify as the input pin for the scan cell’s clock, as shown in Figure 2-4.

![Figure 2-4. Observe Example](image)

- **scan_cell** — An optional replaceable argument that specifies whether DFTAdvisor places a scan cell at the observe test point. Note that these are new scan cells, not scan replacements for existing sequential elements. They are connected into chain(s) during insertion of test logic. If the design contains no scan, the test point scan cells are connected into one or more scan chains, depending on their clock pins. The following switches are the valid options:
-NONE — A switch that specifies not to insert a scan cell at the test point. This is the default.

-INTscan scan_model — A switch and string pair that specifies the DFT library SCANCELL type model that you want DFTAdvisor to insert at the test point. If you use this option, you must first define the scan_model with the Add Cell Models command or the cell_type DFT library attribute. If the test point is for observe, DFTAdvisor uses the output_pin_pathname you specify as the input pin for the scan cell’s clock.

- Lockup lockup_latch_model clock_pin [-INVert | -NOInvert]

If you enable Set Lockup Latch On, DFTAdvisor normally inserts lockup latches where necessary to control timing problems between cells in merged scan chains. This Lockup argument lets you specify for DFTAdvisor to add a lockup latch at any specified location.

If the location (tp_pin_pathname) is a primary output or an instance input pin, the latch is inserted in front of the pin. If the location is a primary input or an instance output pin, the latch is inserted after the pin and will drive all fanouts which were originally driven by the pin.

- lockup_latch_model — A string that specifies the library latch model name. If you use this option, you must first define the model with the Add Cell Models command.

- clock_pin — A string that specifies the pathname of the clock pin to which the clock pin of the latch is connected.

- -INVert | -NOInvert — A switch that specifies whether to insert an inverter between the specified clock_pin and the clock input to the latch. The default is -NOInvert.
Examples

The following example first defines a DFT library model (and2a) of type AND, and then defines a control test point, which DFTAdvisor then generates as part of the Insert Test Logic command:

```
add cell models and2a -type and
add test point /I_6_16/cp control and2a control1
set system mode dft
run
insert test logic
```

Related Commands

- Add Cell Models
- Delete Test Points
- Insert Test Logic
- Report Test Points
- Set Lockup Latch
- Set Test Logic
Add Tied Signals

Scope: Setup mode

Usage

ADD Tied Signals {0 | 1 | X | Z} floating_object_name... [-Pin]

Description

Specifies for DFTAdvisor to hold the named floating objects (nets or pins) at the given state value.

DFTAdvisor creates a tied simulation gate (depending on the tied value) for each tied signal during the design flattening process. If you do not assign a specific value to a floating object, DFTAdvisor ties the object to the default value for all tied objects. You can change the value for tied objects from the invocation default value of unknown (X) by using the Setup Tied Signals command.

Arguments

• **0**
  
  A literal that specifies to tie the floating nets or pins to logic 0 (low to ground).

• **1**
  
  A literal that specifies to tie the floating nets or pins to logic 1 (high to voltage source).

• **X**
  
  A literal that specifies to tie the floating nets or pins to unknown.

• **Z**
  
  A literal that specifies to tie the floating nets or pins to high-impedance

• **floating_object_name**
  
  A required repeatable string that specifies the floating nets or pins to which you want to assign a specific value. The tool assigns the tied value to all floating nets or pins in all modules that have the specified names.

  If you do not specify the -Pin option, the tool assumes the name is a net name. If you do specify the -Pin option, the tool assumes the name is a pin name.
• -Pin
  
  An optional switch specifying that the *floating_object_name* argument that you provide is a floating pin name.

**Examples**

The following example ties all floating signals in the circuit that have the net names vcc and vdd, to logic 1 (tied to high):

```
add tied signals 1 vcc vdd
```

**Related Commands**

- Delete Tied Signals
- Setup Tied Signals
- Report Tied Signals
Add Write Controls

Scope: Setup mode

Usage
ADD WRite Controls {0 | 1} primary_input_pin...

Description
Specifies the off-state value of the write control lines for RAMs.
The Add Write Controls command defines the circuit write control lines and assigns their off-state values. The off-state value of the pins that you specify must be sufficient to keep the RAM contents stable. You may not use clocks, constrained pins, or equivalent pins as write control lines.

Arguments
- 0
  A literal specifying that 0 is the off-state value for the RAM primary_input_pin.
- 1
  A literal specifying that 1 is the off-state value for the RAM primary_input_pin.
- primary_input_pin
  A required repeatable string that specifies the primary input pins that are write control lines for the RAM and to which you want to assign an off-state value.

Examples
The following example assigns an off-state to two write control lines, w1 and w2:
add write controls 0 w1 w2
set system mode dft
run

Related Commands
Delete Write Controls Report Write Controls
Analyze Control Signals

Scope: All modes

Usage

ANAlyze COntrol Signals [-Report_only] [-Verbose]

Description

Identifies and defines the primary inputs of control signals.

The Analyze Control Signals command analyzes each control signal (clocks, set, reset, write-control, read-control, etc.) of every sequential element (DFF, latch, RAM, ROM, etc.) and defines its primary input as a control signal. This analysis also considers pin constraints. The purpose of this analysis is to identify the primary input that needs to be defined as a clock, read-control, or write-control.

The analysis only considers simple combinational gates. If the -Verbose option is specified, the tool issues messages indicating why certain control signals are not identified. At the end of the analysis, statistical information is displayed listing the number of control signals identified, their types, and additional information. By default, all identified control signals are identified and their primary inputs automatically defined as such (i.e., when a clock is identified, an implicit Add Clocks command is performed to define the clock).

![Note] This command performs the flattening process automatically, if executed prior to performing flattening.

Arguments

- -Report_only
  An optional literal that specifies to identify control signals only (does not define the primary inputs as control signals). The invocation default is to automatically define the primary inputs as control signals.

- -Verbose
  An optional literal that specifies to display information on control signals (whether they are identified or not, and why) while the analysis is performed.
Examples

The following example analyzes the control signals, then only provides a verbose report on the control signals in the design. After examining the transcript, you can then perform another analysis of the control signals to add them.

```
analyze control signals -report_only -verbose
```

```
// command: analyze control signals -reports_only -verbose
// ------------------------------------------------------------------------
// Begin control signals identification analysis.
// ------------------------------------------------------------------------
// Warning: Clock line of `/cc01/tim_ccl/addi/post_latch_29/WEITEB_reg/r/ (7352)` is uncontrolledat `/IT12 (4)`.  
... 
... 
// Identified 2 clock control primary inputs.  
//  /IT23 (5) with off-state = 0.  
//  /IT12 (4) with off-state = 0.  
// Identified 0 set control primary inputs.  
// Identified 1 reset control primary inputs.  
//  /IRST (1) with off-state = 0.  
// Identified 0 read control primary inputs.  
// Identified 0 write control primary inputs.  
// ------------------------------------------------------------------------
// Total number of internal lines is 105 (35 clocks, 35 sets, 35 resets,  
// 0 reads, 0 writes).  
// Total number of controlled internal lines is 25 (17 clocks, 0 sets,  
// 8 resets, 0 reads, 0 writes).  
// Total number of uncontrolled internal lines is 80 (18 clocks, 35 sets,  
// 27 resets, 0 reads, 0 writes).  
// Total number of added primary input controls 0 (0 clocks, 0 sets,  
// 0 resets, 0 reads, 0 writes).  
// ------------------------------------------------------------------------
```

```
analyze control signals -verbose
```

### Related Commands

- Add Clocks
- Add Read Controls
- Add Write Controls
- Report Clocks
- Report Read Controls
- Report Write Controls
**Analyze Drc Violation**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: This command operates only after the design rules checker encounters a rule violation.

**Usage**

ANALyze DRC Violation `rule_id-occurrence#`

DFTInsight Menu Path:
- Analyze > Drc Violation...

**Description**

Generates a netlist of the portion of the design involved with the specified rule violation number.

DFTAdvisor stores the results of this command as a netlist that you can view with the optional schematic viewing application (DFTInsight). When you issue the Analyze Drc Violation command, DFTAdvisor includes different simulation data into the netlist depending on the type of rule violation. Even though DFTInsight displays the simulation data, the gate reporting data within the DFTAdvisor session does not change, unless you use the DFTInsight Setup > GateReport >... menu option.

If you have DFTInsight invoked when you issue the Analyze Drc Violation command, the viewer automatically displays the graphical representation of the netlist and marks key instances in the schematic view. Otherwise, in order to view the resulting netlist, you must issue the Open Schematic Viewer command to invoke DFTInsight.

**Arguments**

- `rule_id-occurrence#`

  A required literal and integer argument pair separated by a hyphen that specifies the exact design rule violation (including the occurrence) that you want to analyze. DFTAdvisor traces the violation back to the probable cause and then the schematic viewer displays all the gates in that trace. The argument must include the design rules violation ID (`rule_id`), the specific occurrence
number of that violation, and the hyphen between them. For example, you can analyze the second occurrence of the C3 rule by specifying C3-2. The tool assigns the occurrences of the rules violations as it encounters them, and you cannot change either the rule identification number or the ordering of the specific violations.

The design rule violations and their identification literals divide into five groups: RAM, Clock, Data, Extra, Scannability, and Trace rules violation IDs.

The following lists the RAM rules violation IDs. For a complete description of these violations refer to the “RAM Rules” section in the Design-for-Test Common Resources Manual.

A1 — When all write control lines are at their off-state, all write, set, and reset inputs of RAMS must be at their inactive state.
A2 — A defined scan clock must not propagate to a RAM gate, except for its read lines.
A3 — A write or read control line must not propagate to an address line of a RAM gate.
A4 — A write or read control line must not propagate to a data line of a RAM gate.
A5 — A RAM gate must not propagate to another RAM gate.
A6 — All the write inputs of all RAMs and all read inputs of all data_hold RAMs must be at their off-state during all test procedures, except test_setup.
A7 — When all read control lines are at their off-state, all read inputs of RAMs with the read_off attribute set to hold must be at their inactive state.

The following lists the Clock rules violation IDs. For a complete description of these violations refer to the “Clock Rules” section in the Design-for-Test Common Resources Manual.

C1 — The netlist contains the unstable sequential element in addition to the backtrace cone for each of its clock inputs. The pin data shows the value that the tool simulates when all the clocks are at their off-states and when all the pin constraints are set to their constrained values.
C2 — The netlist contains the failing clock pin and the gates in the path from it to the nearest sequential element (or primary input if there is no
The pin data shows the value that the tool simulates when the failing clock is set to X, all other clocks are at their off-states, and when all pin constraints are set to their constrained values.

**C3** | **C4** — The netlist contains all gates between the source cell and the failing cell, the failing clock and the failing cell, and the failing clock and the source cell. The pin data shows the clock cone data for the failing clock.

**C5** | **C6** — The netlist contains all gates between the failing clock and the failing cell. The pin data shows the clock cone data for the failing clock.

**C7** — The netlist contains all the gates in the backtrace cone of the bad clock input of the failing cell. The pin data shows the constrained values.

**C8** | **C9** — The netlist contains all the gates in the backtrace cone of the failing primary output. The pin data shows the clock cone data for the failing clock.

The following lists the Data rules violation IDs. For a complete description of these violations refer to the “Scan Cell Data Rules” section in the Design-for-Test Common Resources Manual.

**D1** — The netlist contains all the gates in the backtrace cone of the clock inputs of the disturbed scan cell. The pin data shows the pattern values the tool simulated when it encountered the error.

**D2** — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulated for all time periods of the shift procedure.

**D3** — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the master_observe procedure.

**D4** — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the skew_load procedure.

**D5** — The netlist contains the disturbed gate, and there is no pin data.

**D6** | **D7** | **D8** — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the value that the tool simulates when all clocks are at their off-states.
**Analyze Drc Violation Command Dictionary**

**D9** — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the pattern value the tool simulated when it encountered the error.

**D10** — The transparent capture cells in clock procedures must not propagate both old and new data to other state elements.

**D11** — The transparent capture cells in clock procedures must not propagate data to primary outputs.

The following lists the Extra rules violation IDs. For a complete description of these violations refer to the “Extra Rules” section in the *Design-for-Test Common Resources Manual*.

**E2** — There must be no inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell.

**E3** — There must be no inversion between MASTER and SLAVE for any scan cell.

**E4** — Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures.

**E5** — When constrained pins are at their constrained states, and PIs and scan cells are at their specified binary states, X states must not be capable of propagating to an observable point.

**E6** — When constrained pins are at their constrained states, the inputs of a gate must not have sensitizable connectivity to more than one memory element of a scan cell.

**E7** — External bidirectional drivers must be at the high-impedance (Z) state during the application of the test procedure.

**E8** — All masters of all scan cells of a scan chain must use a single shift clock.

**E9** — The drivers of wire gates must not be capable of driving opposing binary values.
The following lists the Scannability rules violation IDs. For a complete description of these violations refer to the “Scannability Rules” section in the Design-for-Test Common Resources Manual:

**S1** — Checks all the clock inputs (including sets and resets) of each nonscan memory element to ensure that these inputs can be turned off.

**S2** — Checks all clock inputs (not including sets and resets) of each nonscan memory element to see whether they can capture data.

**S3** — Checks for mux-DFF style scan to see if defined clocks can be used as shift clocks.

The following lists the Trace rules violation IDs. For a complete description of these violations refer to the “Scan Chain Trace Rules” section in the Design-for-Test Common Resources Manual:

**T2** — The netlist contains the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

**T3** — The netlist contains all the gates in the backtrace cone of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

**T4** — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

**T5 | T6** — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

**T7** — The netlist contains all the gates in the path between the two failing latches. The pin data shows the values the tool simulates for all time periods of the shift procedure.

**T11** — A clock input of the memory element closest to the scan chain input must not be on during the shift procedure prior to the time of the force_sci statement.

**T16** — When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink (PGS) gate in the long path must be at the non-controlling value of the PGS gate.
T17 — Reconvergent pulse generator sink gates cannot connect to any of the following: primary outputs, non-clock inputs of the scan memory elements, ROM gates, non-write inputs of RAMs and transparent latches.

Examples

The following example defines the off-state of a clock incorrectly, causing a C2 rule violation. When a rule violation occurs, you can use the schematic viewer to analyze the probable cause of the error.

With this example, the schematic viewer displays the sequential element associated with the clk input, along with a backward trace through the gates and nets to the associated primary input.

```
add clocks 0 clk
set system mode atpg
// . . .
//  ________________________________
//  Begin scan clock rules checking.
//  ________________________________
//  1 scan clock/set/reset lines have been identified.
//  All scan clocks successfully passed off-state check.
//  Error: Clock /CLK cannot capture data with other clocks off. (C2-1)

open schematic viewer
analyze drc violation c2-1
```

Related Commands

- Open Schematic Viewer
- Set Schematic Display
- Set Drc Handling
Analyze Input Control

Scope: Dft mode

Usage

ANAlalyze INput Control

Description

Specifies for DFTAdvisor to calculate and display the effects of constraining primary input pins to an unknown value on those pins’ control capability.

If you are using partition scan, within any partition, you must constrain to an unknown level those primary inputs of the partition that are uncontrollable at the higher chip-level by using the Add Pin Constraints command. DFTAdvisor then uses those constrained inputs when identifying the non-scan cells that should be in the partition scan chain for that particular partition.

However, because sometimes there is combinational logic between the constrained pin and the sequential element that gets converted to an input partition scan cell, constraining the partition pin can impact the fault detection of this combinational logic.

The Analyze Input Control command determines the controllability factor of a partition pin by removing the X constraint and calculating the controllability improvement on the affected combinational gates. If a partition pin has a significant impact on the fault coverage within that combinational logic, you can consider making that pin a test point or adding a new scan cell either inside or outside of that design partition to effect a higher fault coverage.

DFTAdvisor displays the results of the calculations, with the primary input having the largest controllability gain listed first.

For more information on partition scan, refer to “Understanding Partition Scan” in the Scan and ATPG Process Guide.
Examples

The following example causes DFTAdvisor to perform the calculations and display the results:

```
add pin constraints data1 cx
add pin constraints addr4 cx
add pin constraints addr7 cx
analyze input control
    addr4
    addr7
    data1
```

Related Commands

Add Pin Constraints          Report Testability Analysis
Analyze Output Observe      Setup Scan Identification
**Analyze Output Observe**

Scope: Dft mode

**Usage**

ANAlyze OUtput Observe

**Description**

Specifies for DFTAdvisor to calculate and display the effects on the observability of masked primary output pins.

If you are using partition scan, within each partition you must mask those primary outputs of the partition that are unobservable at the higher chip-level by using the Add Output Masks command. DFTAdvisor then uses those masked outputs when identifying the non-scan cells that should be in the partition scan chain for that partition.

However, there may be combinational logic between the sequential element (which DFTAdvisor converts to an output partition cell) and the primary output, and masking the partition pin may impact the fault detection of this combinational logic.

The Analyze Output Observe command determines the observability factor of a partition pin by removing the mask and calculating the observability improvement on the affected combinational gates. If a partition pin has a significant impact on the fault coverage within that combinational logic, you can consider making that pin a test point or adding a new scan cell either inside or outside of that design partition to enable the higher chip-level fault coverage.

DFTAdvisor displays the results of the calculations, with the primary output that has the largest observability gain listed first.

For more information on partition scan, refer to “Understanding Partition Scan” in the *Scan and ATPG Process Guide*. 
Examples

The following example causes DFTAdvisor to perform the calculations and display the results:

```
add output masks qout1 -hold 1
add output masks addr_res<1> -hold 0
add output masks addr_res<4> -hold 1
analyze output observe
    addr_res<1>
    qout1
    addr_res<4>
```

Related Commands

- Add Output Masks
- Report Testability Analysis
- Analyze Input Control
- Setup Scan Identification
**Analyze Testability**

Scope: Dft mode

**Usage**

`ANAlyze TEstability [-Scoap_only]`

**Description**

Reports general scannability and testability information, along with calculating the controllability and observability values for gates.

The Analyze Testability command reports general scannability and testability information which can help you determine how much partial scan the design may need to achieve high test coverage.

The scannability and testability information reported includes:

- Statistics about the total number of sequential elements, number of scannable sequential elements, number of nonscannable sequential elements, etc.

- Number of scannable sequential elements that need to be scanned to break all global sequential loops

- Number of scannable sequential elements with self loops

- Number of scannable sequential elements required to scan RAM boundaries (if the design contains RAMs)

- Number of scannable sequential elements required to limit sequential depth and consecutive self loops.

  If the design contains sequential loops, the reported sequential depth is estimated.

- Number of uncontrollable and unobservable scannable sequential elements (based on SCOAP analysis)
The information reported is mainly related to the structure of the circuit. If you are using structure-based scan selection, you can use the report to correlate structural criteria with the amount of scan required. For example, you will see a report on how much scan is required to break all sequential loops or to limit sequential depth to a given number. This can help you in determining what parameters to provide to structure-based scan selection.

If you wish to use partial scan, it is recommended that you use the more automated Automatic scan-selection method. The information provided by this report can also give you a measure of how testable the circuit is at any given time, which can help you in determining whether more scan needs to be selected. For example, if all global loops are broken and the sequential depth is small, this indicates that the circuit is likely to achieve high test coverage and that the scan selected thus far may be sufficient.

In addition, this command uses SCOAP testability measures to calculate the controllability and observability of individual gates which can be reported using the Report Testability Analysis command. If you use the -Scoap_only switch, this command only calculates the controllability and observability values.

**Arguments**

- **-Scoap_only**
  A switch that specifies to only compute SCOAP controllability and observability numbers for use with the Report Testability Analysis command. If this switch is not specified, these numbers are still calculated, but in addition, scannability and testability information is calculated and reported.

**Examples**

The following example shows the default output from the Analyze Testability command. The controllability and observability numbers are also calculated, but must be reported using the Report Testability Analysis command (as shown in the next example).

```
DFT> analyze testability
// Number of sequential instances:
// Total = 751
// Scannable = 319 (42.48%)
// Identified = 0 (0.00%)
```
The following example shows the flow of displaying only the controllability values. The report displays the controllability value for the low logic state (where NC means non-controllable), the controllability value for the high logic state, the primitive gate type, the gate identification number, and the pathname to the gate.

```
set system mode dft
...
analyze testability -scoap_only
report testability analysis -control -percent 5
0    NC   TIE0  32   /addr/U15
NC   0    TIE1  53   /addr/U35
1    7001 INV   95   /cntr/U45
5672 1    BUF   382  /blk1/U85
```

**Related Commands**

Add Test Points  
Setup Scan Identification  
Report Testability Analysis
**Close Schematic Viewer**

Tools Supported: DFTAdvisor and DFTInsight  
Scope: All modes

**Usage**

CLOse SChematic Viewer  
DFTInsight Menu Path:  
File > Close

**Description**

Terminates the optional schematic viewing application (DFTInsight).

When you terminate the DFTInsight session, the display netlist remains until you exit the DFTAdvisor session. When you exit DFTAdvisor, it removes the entire $MGC_HOME/tmp/dfti.<process#> directory.

If you change the netlist location with the Set Schematic Display command, DFTAdvisor does not remove the netlist upon exiting. If DFTInsight is still running when you exit the DFTAdvisor session, DFTInsight automatically terminates.

**Examples**

The following example invokes the schematic viewer, creates and displays a netlist, and then terminates the viewing session:

```
open schematic viewer
add display instances i_16_7 -backward -end_point
close schematic viewer
```

**Related Commands**

- Open Schematic Viewer  
- Set Schematic Display
Delete Atpg Constraints

Scope: Bist mode

Prerequisites: You can only delete constraints added with the Add Atpg Constraints command.

Usage

DELete ATpg Constraints *pin_pathname*... | -All

Description

Removes the state restrictions from the specified objects.

The Delete Atpg Constraints command allows you to delete restrictions on pins defined with the Add Atpg Constraints command. During the fault simulation process, the tool adheres to all of the state restrictions that you have defined and not deleted.

Arguments

- *pin_pathname*
  A repeatable string that specifies the pathname of the pin from which you want to remove any ATPG pin constraints.

- -All
  A switch that removes all current, user-defined ATPG constraints from all objects.

Examples

The following example creates two ATPG pin constraints, runs the fault simulation process, and removes all ATPG constraints:

```
set system mode bist
add atpg constraints 0 /corecomp/core_i/phase0
add atpg constraints 1 /corecomp/core_i/phase1
add faults -all
set random patterns 8000
run
delete atpg constraints -all
add atpg constraints 1 /corecomp/core_i/phase0
```
Delete Atpg Constraints

Command Dictionary

```
add atpg constraints 0 /corecomp/core_i/phase1
run
```

**Related Commands**

- Add Atpg Constraints
- Report Atpg Constraints
Delete Buffer Insertion

Scope: All modes

Usage

DELete BUffer Insertion test_pin... | -ALL

Description

Specifies the type of scan test pins on which you want to remove the fanout limit. The default fanout limit on all types of scan test pins is infinity. You can limit the fanout with the Add Buffer Insertion command, and you can remove that limit with the Delete Buffer Insertion command. If you remove the limit, DFTAdvisor resets the limit to the default of infinity.

Arguments

- **test_pin**

  A repeatable literal that specifies the type of the primary input scan pin from which you want DFTAdvisor to remove the fanout limit. The following shows the default pin names for each type of scan pin, but you can change the default names using the Setup Scan Insertion command.

  - **SEN** (scan enable; default name scan_en) — A literal that specifies the primary input pin that enables the scan chain.
  - **SCLK** (scan clock; default name scan_clk) — A literal that specifies the primary input pin that clocks the scan data through the scan chain; the clocked scan type uses this pin.
  - **TEN** (test logic enable; default name test_en) — A literal that specifies the primary input that enables the operation of the test logic circuitry.
  - **TCLK** (test logic clock; default name test_clk) — A literal that specifies the primary input pin that clocks the values DFTAdvisor requires for proper functionality of the test logic.
  - **SMCLK** (master scan clock; default name scan_mclk) — A literal that specifies the primary input that clocks the scan data into the master scan elements of the scan chain when using the LSSD scan type.
Delete Buffer Insertion Command Dictionary

**SSCLK** (slave scan clock; default name scan_sclk) — A literal that specifies the primary input that clocks the scan data into the slave scan elements of the scan chain when using the LSSD scan type.

- **-ALL**
  
  A switch that removes the fanout limits from all the primary input scan pins and returns each to its default setting of infinity.

**Examples**

The following example changes the default settings for test logic and then removes those settings. The two reports show the results of each command.

```
add buffer insertion 5 ten tclk -model buf1a
report buffer insertion
scan_enable <infinity>
scan_clock <infinity>
test_enable 5 buf1a
test_clock 5 buf1a
scan_master_clock <infinity>
scan_slave_clock <infinity>
hold_enable <infinity>
```

```
delete buffer insertion -all
report buffer insertion
scan_enable <infinity>
scan_clock <infinity>
test_enable <infinity>
test_clock <infinity>
scan_master_clock <infinity>
scan_slave_clock <infinity>
hold_enable <infinity>
```

**Related Commands**

Add Buffer Insertion  Report Buffer Insertion
Delete Cell Models

Scope: All modes

Usage

DELete CEll Models \texttt{dftlib\_model...} | {-Type \{INV \mid And \mid Buf \mid OR \mid NAnd \mid NOr \mid Xor \mid INBuf \mid OUtbuf \mid Mux \mid Scancell \mid DFf \mid DLat\}} | -All

Description

Specifies the name of the DFT library cell that DFTAdvisor is to remove from the active list of cells that the user can access when adding test points or that DFTAdvisor can access when inserting test logic.

You originally added the cells to the active list with either the Add Cell Models command or with the \texttt{cell\_type} library attribute. If you remove a cell model from the active list, you only remove the cell from that list and do not change the DFT library.

If you accidentally delete a DFT library cell from the active list with the Delete Cell Models command, you can add the specified cell back into the active list with the Add Cell Models command.

Arguments

- \texttt{dftlib\_model}
  
  A repeatable string that specifies the names of the particular DFT library models that you want DFTAdvisor to remove from the active list.

- \texttt{-Type INV \mid And \mid Buf \mid OR \mid NAnd \mid NOr \mid Xor \mid INBuf \mid OUtbuf \mid Mux \mid Scancell \mid DFf \mid DLat\}}

  A switch and argument pair that specifies the cell model type of all DFT library models that you want DFTAdvisor to remove from the active list. The valid cell model types are as follows:

  - \texttt{INV} — A literal that specifies a one-input inverter gate.
  - \texttt{And} — A literal that specifies a two-input AND gate.
  - \texttt{Buf} — A literal that specifies a one-input buffer gate.
  - \texttt{OR} — A literal that specifies a two-input OR gate.
NAnd — A literal that specifies a two-input NAND gate.

NOr — A literal that specifies a two-input NOR gate.

Xor — A literal that specifies an exclusive OR gate.

INBuf — A literal that specifies a primary input buffer gate.

OUTbuf — A literal that specifies a primary output buffer gate.

Mux — A literal that specify a 2-1 multiplexer.

Scancell — A literal that specify a mux-scan D flip-flop.

DFf — A literal that specify a D flip-flop.

DLat — A literal that specify a D latch.

• -All

A switch that removes all cell models from the active list, including those tagged in the DFT library with the cell_type attribute. This switch does not change the contents of the DFT library, only the active list within DFTAdvisor.

Examples

The following example removes a DFT library model from the active list:

```plaintext
add clocks 0 clk
set test logic -set on -reset on
set system mode dft
add cell models and2 -type and
add cell models or2 -type or
add cell models mux21h -type mux s a b
add cell models nor2 -type nor
delete cell models or2
insert test logic
```

Related Commands

Add Cell Models  Set Test Logic
Report Cell Models
Delete Clock Groups

Scope: Dft mode

Usage

DELete CLock Groups group_name... | -All

Description

Specifies the name of the group that you want to remove from the clock groups list.

If you are going to merge multiple shift clocks together to form one scan chain, you can use the Add Clock Groups command to place the shift clocks in the same clock group. If you make a mistake when defining the clocks within a clock group, you can use the Delete Clock Groups command.

Arguments

- group_name
  A repeatable string that specifies the names of the clock groups that you want to remove. The value of the group_name argument is the same as that which you specified with the Add Clock Groups command.

- -All
  A switch that specifies to remove all the clock groups.

Examples

The following example defines the current clocks, splits those clocks incorrectly into two different groups, removes the clock group that was incorrectly defined, and then continues defining the clock groups:

```
add clock 1 clk1 clk2
add clock 0 pre1 clr1 pre2 clr2
set system mode dft
...
add clock groups group1 clk1 pre2 clr1
delete clock groups group1
add clock groups group1 clk1 pre1 clr1
add clock groups group2 clk2 pre2 clr2
```
Related Commands

Add Clock Groups          Report Clock Groups
Add Clocks               Report Clocks
Delete Clocks

Scope: Setup mode

Usage

DELete CLocks `primary_input_pin... | -All`

Description

Removes primary input pins from the clock list.

The Delete Clocks command removes the specified primary input pins from the clock list. If you remove an equivalence pin from the clock list, DFTAdvisor automatically removes all of the equivalent pins from the clock list.

Arguments

- `primary_input_pin`
  A repeatable string that specifies the list of primary input pins that you want to delete from the clock list.

- `-All`
  A switch that deletes all pins from the clock list.

Examples

The following example deletes an incorrect clock from the clock list:

```
add clocks 1 clock1
add clocks 1 clock2
delete clocks clock1
```

Related Commands

Add Clocks
Report Clocks
Delete Display Instances

Tools Supported: DFTAdvisor and DFTInsight
Scope: Bist and Dft modes

Usage

DELete DIspay Instances \{gate_id# | instance_name\}... | -All

DFTInsight Menu Item:
Display > Deletions > All | Selected

Description

Removes the specified objects from display in DFTInsight.
The Delete Display Instance command removes the specified instance from the Schematic View area of the DFTInsight window. When you remove objects, DFTInsight automatically updates the schematic view window with the new display.

DFTInsight marks key instances in the schematic view. If you delete a key instance, the command removes the marked instance from the schematic view and updates the marked instances list. The list is updated so that if you add the instance later via another command, DFTInsight will not display it as marked.

Arguments

- gate_id#  
  A repeatable integer that specifies the gate identifications of the gates that you want to remove from the DFTInsight display. The value of the gate_id# argument is the unique identification number that DFTAdvisor automatically assigns to every gate within the design during the model flattening process.

- instance_name  
  A repeatable string that specifies the names of the top-level instances within the design that you want to remove from the DFTInsight display.

- -All  
  A switch that specifies to remove all the objects in the current DFTInsight display, leaving a blank display window. When you use this switch, even
though the display becomes blank, DFTInsight does not delete the actual display netlist.

Examples
The following example first causes DFTInsight to display three gates, and then removes one of the gates from the graphical display:

```
add display instances 32 i_2_16 62
delete display instances 62
```

This command removes the remaining gates leaving the display window blank:

```
delete display instances -all
```

Related Commands

Add Display Instances Report Display Instances
Delete Faults

Scope: Bist mode

Prerequisites: You must add faults with the Add Faults command before you can delete them.

Usage

DELete FAults {object_pathname... | -All} [-Stuck_at {01 | 0 | 1}] [-Untestable]

Description

Removes faults from the current fault list.

The Delete Faults command deletes faults from the fault list added using the Add Faults command.

You can optionally specify faults with a specific stuck-at value. If you do not specify a stuck-at value when deleting a fault, the command deletes both the “stuck-at-0” and “stuck-at-1” faults from the fault list.

When you issue this command, DFTAdvisor discards all patterns in the current test pattern.

In addition to specifying faults with a specific stuck-at value, DFTAdvisor allows you to specify faults that are untestable. Untestable faults are common when using BIST techniques or random patterns. This includes faults that the tool cannot detect due to either constraints or the use of a single capture clock. This also includes faults on circuitry that does not have a scan propagable path to a Multiple Input Signature Register (MISR). You can specify the -Untestable switch to remove these fault types.

Arguments

• object_pathname
  A repeatable string that specifies a list of pins or instances.

• -All
  A switch that deletes all faults in the current fault list.
- **Stuck\_at 01 | 1 | 0**
  
  An optional switch and literal pair that specifies the stuck-at values that you want to delete. The valid stuck-at literals are as follows:
  
  01 — A literal that deletes both of the “stuck-at-0” and “stuck-at-1” faults. This is the default.
  
  0 — A literal that deletes only the “stuck-at-0” faults.
  
  1 — A literal that deletes only the “stuck-at-1” faults.

- **-Untestable**
  
  An optional switch that deletes all untestable faults.

**Examples**

The following example deletes a stuck-at-0 fault from the current fault list after adding all the faults to the circuit, but before performing a fault simulation run:

```
set system mode bist
add faults -all
delete faults i_1006/i1 -stuck_at 0
run
```

**Related Commands**

- **Add Faults**
- **Report Faults**
Delete LFSR Connections

Scope: Setup mode

Prerequisites: You must define LFSR connections with the Add LFSR Connections command before you can delete them. This command intended for an LBISTArchitect design flow.

Usage

DELETE LFsr Connections primary_pin... | -All

Description

Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).

The Delete LFSR Connections command deletes the connections between the LFSRs and the primary pins specified with the Add LFSR Connections command. You can use the Report LFSR Connections command to display all the current connections between LFSRs and primary pins.

Arguments

- primary_pin
  A repeatable string that lists the primary pins whose connections to LFSRs you want to delete.

- -All
  A switch that deletes all of the connections between LFSRs and primary pins.

Examples

The following example changes the definition of an LFSR connection by deleting it and then re-adding it with a new definition:

```plaintext
add lfsrs lfsr1 prpg 5 15 -serial -in
add lfsr taps lfsr1 2 3 4
add lfsr connections scan_in.1 lfsr1 2
delete lfsr connections scan_in.1
add lfsr connections scan_in.2 lfsr1 2
add lfsrs lfsr1 prpg 5 15 -serial -in
```

DELete LFsr Connections primary_pin... | -All
Related Commands

Add LFSR Connections   Report LFSR Connections
Delete LFSR Taps

Scope: Setup mode

Prerequisites: You must add LFSR taps with the Add LFSR Taps command before you can delete them. This command intended for an LBISTArchitect design flow.

Usage
DELete LFsr Taps lfsr_name {tap_position... | -All}

Description
Removes the tap positions from a Linear Feedback Shift Register (LFSR).

The Delete LFSR Taps command deletes the specified LFSR tap positions added with the Add LFSR Taps command. You can display the current tap positions of all defined LFSRs by using the Report LFSRs command.

Arguments
• lfsr_name
  A string that specifies the reference name of the LFSR whose tap positions you want to delete.

• tap_position
  A repeatable string that specifies the list of tap positions that you want to delete from the lfsr_name.

• -All
  A switch that deletes all of the tap positions from the lfsr_name.

Examples
The following example changes an LFSR tap position by deleting it and then adding a new tap position:

  add lfsrs lfsr1 prpg 5 15 -serial -in
  add lfsrs lfsr2 Prpg 5 13 -serial -in
  add lfsr taps lfsr1 2 3 4
  add lfsr taps lfsr2 1 3
delete lfsr taps lfsr1 3
add lfsr taps lfsr1 1

Related Commands

Add LFSR Taps
Report LFSRs

Setup LFSRs
Delete LFSRs

Scope: Setup mode

Prerequisites: You must define LFSRs with the Add LFSRs command before you can delete them. This command intended for an LBISTArchitect design flow.

Usage

DELete LFsr lfsr_name... -All

Description

Removes the specified Linear Feedback Shift Registers (LFSRs).

The Delete LFSRs command deletes LFSRs defined with the Add LFSRs command. You can use the Report LFSRs command to display a list of the current LFSRs with their current values and tap positions. When you delete an LFSR, the tool also deletes all its taps and pin connections.

Arguments

- \textit{lfsr\_name}
  
  A repeatable string that specifies the reference names of the LFSRs which you want to remove.

- \textit{-All}
  
  A switch that deletes all defined LFSRs.

Examples

The following example changes the definition of an LFSR by deleting it and then re-adding it with a new definition:

\begin{verbatim}
add lfsrs lfsr1 prpg 5 15 -serial -in
add lfsrs lfsr2 prpg 5 13 -serial -in
add lfsrs lfsr3 prpg 5 11 -parallel -out
delete lfsrs lfsr3
add lfsrs lfsr3 prpg 5 11 -parallel -in
\end{verbatim}
**Related Commands**

- Add LFSRs
- Report LFSRs
- Setup LFSRs
Delete Mapping Definition

Scope: Setup and DFT modes

Usage

DELete MApping Definition -All | {object_name [-Instance | -Module]
 [-Nonscan_model nonscan_model_name] [-Scan_model scan_model_name]
 [-Output [scan_output_pin_name]]}

Description

Returns the nonscan to scan model mapping to the mapping defined by
DFTAdvisor.

The Delete Mapping Definition command deletes the mapping of nonscan models
to scan models. You can remove the scan model mapping for an individual
instance, all instances under a hierarchical instance, all instances in all
occurrences of a module in the design, or all occurrences of the model in the entire
design. Additionally, you can remove the mapping the scan output pin of the scan
model in the same manner.

To return the scan mapping back to the library default, you can specify the
nonscan model, then the scan mapping and output mapping are removed from the
model. If you specify both the nonscan and scan model, then the scan and output
mapping is removed for only those instances that match the nonscan and scan
model.

When only removing the scan output pin mapping, you specify the scan model
(and not the nonscan model). If you also specify the output scan pin, then only
scan candidates matching the scan model and output pin have their output pin
mapping removed.

Arguments

- -All

A required switch that specifies to remove all scan and output mapping in the
entire design.
• **object_name**
  A required string that specifies the name of the nonscan model you want to remove the mapping. You can also specify an instance, hierarchical instance, module, or scan model.

  o If this argument is the name of an instance or hierarchical instance, the -Instance switch is required and the model must be specified with the -Nonscan_model switch or -Scan_model switch.

  o If this argument is the name of a module, then the -Module switch is required and the model must be specified with the -Nonscan_model or -Scan_model switch.

  o If this argument is a scan model, then the -Output switch is required. Since you specified a scan model, you can only remove the scan output pin mapping.

• **-Instance | -Module**
  An optional switch that specifies the type of the object_name argument. If neither switch is specified, the object_name is a model (the default).

    o If you specify -Instance and the instance is primitive, then only the named instance has its mapping changed.

    o If you specify -Instance and the instance is hierarchical, then all instances under that instance matching the -Nonscan_model or (for output mapping) matching the -Scan_model have their mapping changed.

    o If you specify -Module, then for all occurrences of that module, all instances within that module that match the -Nonscan_model or (for output mapping) matching the -Scan_model have their mapping changed.

• **-Nonscan_model nonscan_model_name**
  A switch and string pair that specifies the name of the nonscan model that you want to remove the scan and pin mapping. This argument is required only if
you specify -Instance or -Module switch, otherwise, you can specify the
nonscan model in the object_name argument.

- **-Scan_model scan_model_name**
  A switch and string pair that specifies the name of the scan model that is
  mapped to the specified nonscan model. This argument is required only if you
  want to constrain the removing of the scan mapping or are just removing the
  scan output pin mapping based on -Instance or -Module.

- **-Output [scan_output_pin_name]**
  An optional switch and optional string pair that specifies to remove the scan
  output pin. Specifying just the -Output switch removes all changed scan output
  pins for the specified scan model, while specifying the switch with a pin name
  removes the mapping for only scan models that use that pin for the scan output.

**Examples**

The following example removes the scan and output mapping for all occurrences
of the fd1 nonscan model in the design:

```
deleter mapping definition fd1
```

The following example removes the scan and output mapping for each occurrence
of the fd1 nonscan model that is mapped to the fd1s scan model and has the scan
output pin mapped to “qn”:

```
deleter mapping definition fd1 -scan_model fd1s -output qn
```

The following example removes the scan and output mapping for each occurrence
of the fd1 nonscan model under the hierarchical instance “/top/counter1”:

```
deleter mapping definition /top/counter1 -instance -nonscan_model fd1
```

The following example removes the scan and output mapping for each occurrence
the fd1 nonscan model that is mapped to the fd1s2 scan model in the “counter”
module and for all occurrences of that module in the design:

```
deleter mapping definition counter -module -nonscan_model fd1 -scan_model fd1s2
```

The following example removes the scan output pin mapping and returns it to the
library default for all occurrences of the fd1s scan model in the design:

```
deleter mapping definition fd1s -output
```
The following example removes the scan output pin mapping and returns it to the library default for all occurrences of the fd1s scan model in the design with the scan output pin set to “qn”:

```
delete mapping definition fd1s -output qn
```

**Related Commands**

- Add Mapping Definition
- Report Mapping Definition
Delete Mos Direction

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

DELETE MOS Direction subckt_name instance_name

Description

Removes the assigned direction of a MOS transistor.

The Delete Mos Direction command removes the direction of a MOS transistor in the Spice design or library which was assigned with the Add Mos Direction command. This command makes the transistor bi-directional again, and therefore must be defined again with the Add Mos Direction command.

Arguments

- subckt_name
  A required string that specifies the name of the SUBCKT that contains the instance for which you are removing the direction.

- instance_name
  A required string that specifies the name of the instance within the SUBCKT for which you are removing the direction.

Examples

The following example removes the direction of the instance (K5) MOS transistor of the subckt FADD2:

   delete mos direction FADD2 K5

Related Commands

Add Mos Direction   Report Mos Direction
Extract Subckts
Delete Mtpi Controller

Scope: Setup mode

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

DELete MTpi Controller controller_name | -All

Description

Deletes the MTPI controller(s).

The Delete Mtpi Controller command deletes the specified or all MTPI controllers defined with the Add Mtpi Controller command. All output data associated with the controller is also deleted.

Arguments

- controller_name
  A string that specifies the name of the MTPI controller to delete.

- -All
  A switch that specifies to delete all MTPI controllers.

Examples

The following example deletes the “controller1” MPTI controller:

```
add mtpi controller controller1 /corecomp/core_i/phase_1
  /corecomp/core_i/phase_2 /corecomp/core_i/phase_3
delete mtpi controller controller1
```

Related Commands

Add Mtpi Controller      Report Mtpi Controller
Add Mtpi Output          Set Bist Initialization
Delete Mtpi Output
Delete Mtpi Output

Scope: All modes

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

-All | {controller_name} {-All | cycle_number...}

Description

Deletes the MTPI controller output definitions.

The Delete Mtpi Output command deletes the output data definition for the specified or all MTPI controllers. If you specify an individual controller, you may also specify for which cycle to delete the output data.

Arguments

- **-All**
  A switch that specifies to delete the output data of all MTPI controllers.

- **controller_name** {-All | cycle_number...}
  A string that specifies the name of the MTPI controller whose output data you want to delete.

  - **-All** — A switch that specifies to delete all output data for the specified MTPI controller.

  - **cycle_number** — A repeatable integer that specifies the number of the cycle whose output data you want to delete.

Examples

The following example adds output data for the controller1 controller, deletes the output data of the controller for the second and ninth cycles, and then enters new values:

```
add mtpi output controller1 0 001
add mtpi output controller1 2 010
add mtpi output controller1 9 100
add mtpi output controller1 18 000
```
delete mtpi output controller1 2 9

add mtpi output controller1 2 100
add mtpi output controller1 9 010

Related Commands

Add Mtpi Controller  Report Mtpi Controller
Add Mtpi Output    Set Bist Initialization
Delete Mtpi Controller
Delete Net Property

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

DELete NET Property \{net_name \{-VDD | -GND\}\} \-All

Description

Resets the VDD or GND net property in the Spice design and library.

The Delete Net Property command resets the specified VDD or GND net property in the Spice design and Spice library so that the net is no longer considered as VDD or GND.

Arguments

- \textit{net\_name}
  A required string that specifies the name of the net which you want to reset from VDD or GND.

- \textit{-VDD \| -GND}
  A required switch that specifies whether the net is VDD or GND.

- \textit{-All}
  A required switch that specifies to delete all net properties regardless of the type of net.

Examples

The following example resets the ZGND net from GND in the loaded Spice design and Spice library.

\texttt{delete net property ZGND -gnd}

Related Commands

Add Net Property \hspace{2cm} Report Net Properties
Delete Nofaults

Scope: Setup mode

Usage

DELete NOfaults *pathname* ... |-All [-Instance | -Module] [-Stuck_at {01 | 0 | 1}]

Description

Removes the no-fault settings from either the specified pin or instance pathnames.

The Delete Nofaults command deletes the nofault settings which were previously specified with the Add Nofaults command. You can optionally specify nofault settings that have a specific stuck-at value. If you do not specify a stuck-at value when deleting a nofault setting, the command deletes both the “stuck-at-0” and “stuck-at-1” nofault settings.

If the pathname is a pin, then DFTAdvisor removes the nofault on only that pin. If the pathname is an instance, then the tool removes all pin nofaults on the top-level of that instance, along with all the pin faults underneath that instance (if it is a hierarchical instance). If the pathname is a module, then the tool removes all pin nofaults on the top-level of the module, along with all the pin nofaults on all instances and pins underneath that module for every occurrence of that module in the design.

You can use the Report Nofaults command to display all the current nofault settings.

Arguments

- **pathname**
  
  A repeatable string that specifies the pin pathnames or the instance pathnames from which you want to delete the nofault settings. If you specify an instance or module pathname, you must also specify the -Instance or -Module switch.

- **-All**
  
  A switch that specifies to delete the nofault settings on all pin pathnames or instance pathnames. The default is pin pathnames unless you specify the -Instance or -Module switch.
• -Instance | -Module
  An optional switch that specifies to interpret the pathname argument and the
  -All switch (if specified) as instance or module pathnames. If the pathname is a
  module, all instances and pins within the module are affected. If neither switch
  is specified, the default pathname is a pin pathname.

• -Stuck_at 01 | 0 | 1
  An optional switch and literal pair that specifies the stuck-at values which you
  want to delete. The valid stuck-at literals are as follows:

  01 — A literal that specifies to delete both the “stuck-at-0” and “stuck-at-1”
       nofault settings. This is the default.

  0 — A literal that specifies to only delete the “stuck-at-0” nofault settings.

  1 — A literal that specifies to only delete the “stuck-at-1” nofault settings.

Examples

The following example will delete an extra added no fault instance.

```
add nofaults i_1006 i_1007 i_1008 -instance
report nofaults
USER  : 01  i_1006/IN
USER  : 01  i_1006/OUT
USER  : 01  i_1007/IN
USER  : 01  i_1007/OUT
USER  : 01  i_1008/IN
USER  : 01  i_1008/OUT

delete nofaults i_1007 -instance
report nofaults
USER  : 01  i_1006/IN
USER  : 01  i_1006/OUT
USER  : 01  i_1008/IN
USER  : 01  i_1008/OUT
```

Related Commands

Add Nofaults  Report Nofaults
Delete Nonscan Instances

Scope: All modes

Usage

DELete NONscan Instances \{pathname... [-INStance | -Control_signal | -Module]} | -All\} [-Class \{User | System | Full\}]

Description

Removes the specified sequential instances from the non-scan instance list. The Delete Nonscan Instances command deletes sequential instances, instances specified by control signals, or all instances within the specified module that were previously added to the non-scan instance list by using either the Add Nonscan Instances command or the Dont_touch property in a Genie netlist. You can delete either a specific list of instance names or all instances.

If these nonscan instances are ignored for scannability checks and then the Delete Nonscan Instances command is entered in Dft mode, these specified instances will not be eligible for scan. You must go back to the Setup mode and then re-enter Dft mode to have scannability checks performed on these instances which can make them eligible for scan.

To display the current non-scan instance list use the Report Nonscan Instances command.

Arguments

- \textit{pathname}
  
  A repeatable string that specifies either the pathnames of the instances or signals that control instances that you want DFTAdvisor to delete from the non-scan instance list.

- \textit{-INStance} | -Control_signal | -Module
  
  A switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, ...)” where clkgen is the module name. You can only use the -Control_signal option in Dft mode. The default is -Instance.
Delete Nonscan Instances Command Dictionary

- **-All**
  A switch that specifies to delete all instances from the non-scan instance list.

- **-Class User | System | Full**
  An optional switch and literal pair that specifies the source (or class) of the non-scan instance which you want to delete. The valid literals are as follows:

  - **User** — A literal that specifies to only delete the non-scan instances entered by the user using the Add Nonscan Instances command. This is the default.
  - **System** — A literal that specifies to only delete the non-scan instances described in the Genie netlist with the Dont_touch property.
  - **Full** — A literal that specifies to delete all the non-scan instances in the user and system class.

**Examples**

The following example deletes an extra sequential non-scan instance called i_1007, then performs a full scan identification run thereby allowing DFTAdvisor to treat the non-scan instance i_1007 as a scan cell during the identification process:

```
set system mode dft
add nonscan instances i_1006 i_1007 i_1008
delete nonscan instances i_1007
setup scan identification full_scan
run
```

**Related Commands**

- **Add Nonscan Instances**
- **Report Nonscan Instances**
- **Setup Scan Identification**
- **Set Nonscan Handling**
Delete Nonscan Models

Scope: All modes

Usage

DELete NONscan Models *model_name*... | -All [-Class {User | System | Full}]

Description

Removes from the non-scan model list the specified sequential DFT library models.

When you issue the Add Nonscan Models command on an DFT library model, DFTAdvisor places all instances of that DFT library model into the user-specified non-scan instance list. Once you remove a model from the non-scan model list with the Delete Nonscan Models command, DFTAdvisor then has the freedom to decide whether to place those instances of that model in the scan instance list.

If these nonscan instances (models) are ignored for scannability checks and then the Delete Nonscan Models command is entered in Dft mode, these specified instances (models) will not be eligible for scan. You must go back to the Setup mode and then re-enter Dft mode to have scannability checks performed on these instances (models) which can make them eligible for scan.

DFTAdvisor decides whether to place individual instances in the scan instance list based on many parameters including the scan setup settings. For example, if the scan setup has been changed to All with the Setup Scan Insertion command, then DFTAdvisor is forced to place all available sequential instances into the scan instance list.

Arguments

- *model_name*
  
  A repeatable string that specifies the model names that you want to delete from the non-scan model list. Enter the model names as they appear in the DFT library.

- -All
  
  A switch that specifies to delete all models from the non-scan model list.
• -Class User | System | Full

An optional switch and literal pair that specifies the class code of the non-scan model that you specify. The valid literal names are as follows:

User — A literal that specifies that the list of non-scan models were previously added by using the Add Nonscan Models command. This is the default class.

System — A literal that specifies that the list of non-scan models were added by DFTAdvisor.

Full — A literal that specifies that the list of non-scan models consist of both the user and system class.

Examples

The following example deletes an extra sequential non-scan model called d_flip_flop2, then performs a full scan identification run thereby allowing DFTAdvisor to treat the non-scan model d_flip_flop2 as a scan cell during the identification process:

```
set system mode dft
add nonscan models d_flip_flop1 d_flip_flop2
delete nonscan models d_flip_flop2
setup scan identification full_scan
run
```

Related Commands

Add Nonscan Instances
Add Nonscan Models
Report Nonscan Models
Set Nonscan Handling
Delete Notest Points

Scope: DFT mode

Prerequisites: You must add circuit points with the Add Notest Points command before you can delete them.

Usage

DELeete NOtest Points {pin_pathname... | instance_pathname...} | -ALL | {Path critical_pathname} | -ALL_Paths

Description

Removes the specified pins from the list of notest points which the tool cannot use for testability insertion.

The Delete Notest Points command deletes the definition of pins, all pins within a instance, or paths that you have previously added using the Add Notest Points command. These notest circuit points identify output pins of cells or paths within the circuit that DFTAdvisor is not to use for insertion of controllability and observability circuitry. You can display a list of these current circuit points and their associated pins by using the Report Notest Points command.

When you delete a critical path, the critical path is removed from the list of active paths (paths read in using Add Notest Points). For each gate in a removed path (which is in no other remaining active paths), the no test point restriction is removed.

You use this command primarily for implementing BIST circuitry.

Arguments

- pin_pathname
  A repeatable string that specifies a list of pins for which you want to delete the circuit points that DFTAdvisor cannot use for testability insertion.

- instance_pathname
  A required repeatable string that lists the instances whose output pins you want to delete the circuit points that DFTAdvisor cannot use for testability insertion. All output pins within that (hierarchical) instance are removed from the list of pins that should be excluded from consideration.
Delete Notest Points Command Dictionary

- **-ALL**
  A switch that deletes all previously-added circuit points and critical paths.

- **-Path critical_pathname**
  A required switch and name pair that specifies to delete the named critical path. You can list the names of the critical paths using the Report Notest Points command with the -Paths switch. For more information on the format of the file, refer to “The Path Definition File” in the Scan and ATPG Process Guide.

- **-ALL_Paths**
  A required switch that specifies to delete all critical paths.

**Examples**

The following example deletes an incorrect notest circuit point and corrects it with a new circuit point before performing testability analysis:

```plaintext
set system mode dft
add notest points tr_i ts_i
delete notest points tr_i
add notest points tr_io
```

**Related Commands**

Add Notest Points  Report Notest Points
Delete Output Masks

Scope: Setup mode

Usage

DELete OUtput Masks *primary_output*... | -All

Description

Removes the masking of the specified primary output pins.

DFTAdvisor uses primary output pins as the observe points during the scan identification process. When you mask a primary output pin with the Add Output Masks command, DFTAdvisor marks that pin as an unobservable primary output during the identification process.

You can set a default mask for all output and bi-directional pins using the Setup Output Masks command. You can add a hold value to a default mask with the Add Output Masks command, or remove a hold value using the Delete Output Masks command. To turn off the default masks for all output pins, you must use the Setup Output Masks command with the Off literal.

Arguments

- *primary_output*
  
  A repeatable string that specifies the names of the primary output pins that you want to unmask.

- -All
  
  A switch that specifies to unmask all primary outputs that you previously masked by using the Add Output Masks command.

Examples

The following example first incorrectly chooses two of the design’s primary output pins to mask. The example then un masks the one primary output that was inappropriate, masks the correct primary output, and then displays the complete list of primary output pins that are currently masked from being used as observation points.
Delete Output Masks

Command Dictionary

```
add output masks q1 qb3 -hold 1
delete output masks qb3
add output masks q1 -hold 0
report output masks
    q1  hold1
    qb1 hold0
```

**Related Commands**

- Add Output Masks
- Analyze Output Observe
- Report Output Masks
- Setup Output Masks
Delete Pin Constraints

Scope: Setup mode

Usage

DELete PIn Constraints *primary_input_pin*... | -All

Description

Removes the pin constraints from the specified primary input pins.

The Delete Pin Constraints command deletes pin constraints that were previously added to the primary inputs with the Add Pin Constraints command. You can delete the pin constraints for specific pins or for all pins.

Note

This command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

You can set a default pin constraint for all input and bi-directional pins using the Setup Pin Constraints command. The pin constraints set by the Setup Output Masks command can have their values overridden with the Add Pin Constraints command. You can remove an override of a default pin constraint using the Delete Pin Constraints command. To remove the default pin constraint for all input pins, you should use the Setup Pin Constraints command with the None literal.

Arguments

- *primary_input_pin*
  
  A repeatable string that specifies a list of primary input pins whose pin constraints you want to delete.

- -All
  
  A switch that specifies to delete the pin constraints of all primary input pins.
Delete Pin Constraints

Command Dictionary

Examples

The following example adds two pin constraints and then deletes one of them:

```
add pin constraints ph1 c0
add pin constraints ph2 c0
delete pin constraints ph1
```

Related Commands

- Add Pin Constraints
- Setup Pin Constraints
- Report Pin Constraints
Delete Pin Equivalences

Scope: Setup mode

Usage
DELete PIN Equivalences primary_input_pin... | -All

Description
Removes the pin equivalence specifications for the designated primary input pins.

The Delete Pin Equivalences command deletes the equivalence specifications that were previously added to the primary inputs with the Add Pin Equivalences command. You can delete pin equivalences for specific pins or for all pins.

Note
This command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

Arguments
- primary_input_pin
  A repeatable string that specifies a list of primary input pins whose equivalence specifications you want to delete.
- -All
  A switch that specifies to delete all pin equivalence effects.

Examples
The following example deletes an incorrect pin equivalence specification and adds the correct one:

  add pin equivalences indata2 -invert indata4
  delete pin equivalences indata2
  add pin equivalences indata3 -invert indata4

Related Commands
Add Pin Equivalences
Report Pin Equivalences
Delete Primary Inputs

Scope: Setup mode

Usage

DELete PRimary Inputs {net_pathname... | primary_input_pin... | -All} [-Class {User | System | Full}]

Description

Removes the specified primary inputs from the current netlist.

The Delete Primary Inputs command deletes the primary inputs that you specify from the circuit. You can delete either the user class, system class, or full classes of primary inputs. If you do not specify a class, the tool deletes the primary inputs from the user class.

You can display a list of any class of primary inputs by using the Report Primary Inputs command.

Arguments

- `net_pathname`
  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- `primary_input_pin...`
  A repeatable string that specifies a list of primary input pins that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- `-All`
  A switch that deletes all primary inputs. You can specify the class of primary inputs to delete with the -Class switch.
- **Class** `User` | `System` | `Full`

An optional switch and literal pair that specifies the class code of the designated primary input pins. The valid class code literal names are as follows:

- **`User`** — A literal specifying that the primary inputs were added using the Add Primary Inputs command. This is the default class.

- **`System`** — A literal specifying that the primary inputs derive from the netlist.

- **`Full`** — A literal specifying that the primary inputs consists of both user and system classes.

**Examples**

The following example deletes an extra added primary input from the user class of primary inputs:

```
add primary inputs indata2 indata4 indata6
delete primary inputs indata4 -class user
```

**Related Commands**

- Add Primary Inputs
- Report Primary Inputs
- Write Primary Inputs
Delete Primary Outputs

Scope: Setup mode

Usage

DELete PRimary Outputs {net_pathname... | primary_output_pin... | -All} 
[-Class {User | System | Full}]

Description

Removes the specified primary outputs from the current netlist.

The Delete Primary Outputs command deletes the primary outputs that you specify from the circuit. You can delete either the user class, system class, or full classes of primary outputs. If you do not specify a class, the tool deletes the primary outputs from the user class.

You can display a list of any class of primary outputs by using the Report Primary Outputs command.

Arguments

- net_pathname
  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- primary_output_pin
  A repeatable string that specifies a list of primary output pins that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- -All
  A switch that deletes all primary outputs. You can specify the class of primary outputs to delete with the -Class switch.
-Class User | System | Full

An optional switch and literal pair that specifies the class code of the primary output pins that you specify. The valid literal names are as follows:

User — A literal specifying that the list of primary outputs were added using the Add Primary Outputs command. This is the default class.

System — A literal specifying that the list of primary outputs derive from the netlist.

Full — A literal specifying that the list of primary outputs consists of both the user and system class.

Examples

The following example deletes a primary output from the system class of primary outputs:

```
delete primary outputs outdata1 -class system
```

Related Commands

Add Primary Outputs  Write Primary Outputs
Report Primary Outputs
Delete Read Controls

Scope: Setup mode

Usage
DELete REad Controls primary_input_pin... | -All

Description
Removes the read control line off-state definitions from the specified primary input pins.

The Delete Read Controls command deletes the off-state definition of the read control lines previously defined with the Add Read Controls command. You can delete the read control line definitions for specific pins or for all pins.

Arguments

- primary_input_pin
  A repeatable string that specifies a list of primary input pins from which you want to delete any read control line off-state definitions.

- -All
  A switch that specifies to delete the read control line off-state definitions for all primary input pins.

Examples
The following example removes an incorrect read control line off-state definition, and then creates the correct off-state for that read control line:

- `add clocks 0 clk`
- `add read controls 0 r1 r2`
- `delete read controls r1`
- `add read controls 1 r1`
- `set system mode dft`

Related Commands
Add Read Controls  Report Read Controls
Delete Scan Chains

Scope: Setup

Usage

DELETE SCAN CHAINS chain_name... | -All

Description

Removes the specified scan chain definitions from the scan chain list.

The Delete Scan Chains command deletes scan chains previously defined with the Add Scan Chains command. You can delete the definitions of specific scan chains or of all scan chains.

When you remove a scan chain definition, it is only the definition you are removing, not the scan chain itself. If you need to remove the scan chain itself, you can use the Ripup Scan Chains command.

Arguments

- chain_name
  A repeatable string that specifies the names of the scan chain definitions that you want to delete.

- -All
  A switch that specifies to delete all scan chain definitions.

Examples

The following example defines several scan chains, adding them to the scan chain list, then deletes one of the scan chains:

```plaintext
add scan chains chain1 group1 indata2 outdata4
add scan chains chain2 group1 indata3 outdata5
add scan chains chain3 group1 indata4 outdata6
delete scan chains chain2
```

Related Commands

Add Scan Chains
Report Scan Chains
Ripup Scan Chains
Delete Scan Groups

Scope: Setup mode

Usage
DELete SCan Groups group_name... | -All

Description
Removes the specified scan chain group definitions from the scan chain group list.

The Delete Scan Groups command deletes scan chain groups previously defined with the Add Scan Groups command. You can delete the definitions of specific scan chain groups or of all scan chain groups.

When you delete a scan chain group, the tool also deletes all scan chains within the group.

Arguments
• group_name
  A repeatable string that specifies the names of the scan chain group definitions that you want to delete.

• -All
  A switch that specifies to delete all the scan chain group definitions, which also automatically causes DFTAdvisor to remove all scan chain definitions.

Examples
The following example defines two scan chain groups, adding them to the scan chain group list, then deletes one of the scan chain groups:

  add scan groups group1 scanfile1
  add scan groups group2 scanfile2
  delete scan groups group1

Related Commands
Add Scan Groups Report Scan Groups
Delete Scan Instances

Scope: All modes

Usage

DELe te SCan Instances \{pathname... [-INStance | -Control_signal | -Module]} | -All

Description

Removes the specified sequential instances from the user-identified scan instance list.

The Delete Scan Instances command deletes sequential instances, instances specified by control signals, or all instances within the specified module that were previously added to the scan instance list by using the Add Scan Instances command. You can delete either a specific list of instance names or all instances.

User-identified scan instances result from using the Add Scan Instances or Add Scan Models commands. DFTAdvisor also selects the sequential instances by using the identification type you specify with the Setup Scan Identification command (system-identified).

If you issue a Run command after removing an instance from the user-identified scan list with the Delete Scan Instances command, DFTAdvisor then has the option of including it in the system-identified scan instance list.

Arguments

- **pathname**
  A repeatable string that specifies the pathnames of the instances or control signals (that control instances) which you want DFTAdvisor to delete from the user-identified scan instance list. The pathnames must be user-identified scan instances or control signals (that control instances) which you previously selected with the Add Scan Instances command.

- **-INStance | -Control_signal | -Module**
  A switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, ...)” where clkgen is the module name. You can only use the -Control_signal option in Dft mode. The default is -Instance.
Delete Scan Instances

- **-All**
  A switch that specifies to delete all instances from the user-identified scan instance list. This switch does not affect the instances in the system-identified scan instance list.

**Examples**

The following example deletes an extra sequential scan instance that was defined to be treated as a scan cell; thus, the deleted instance is no longer included in the user-identified scan instance list:

```
set system mode dft
add scan instances i_1006 i_1007 i_1008
delete scan instances i_1007
```

**Related Commands**

- Add Scan Instances
- Report Scan Instances
Delete Scan Models

Scope: All modes

Usage

DELete SCan Models model_name... | -All

Description

Removes the specified sequential models from the scan model list.

The Delete Scan Models command deletes all instances of the sequential models that you specify. This includes removing all instances of the model_name from the user-identified scan instance list. You can delete a specific list of sequential models or all the models.

There are two ways that sequential instances can be identified for scan (system and user). You can explicitly identify scan instances with either with the Add Scan Instances or the Add Scan Models commands (user-identified). DFTAdvisor also selects the sequential instances by using the identification type you specify with the Setup Scan Identification command. The Delete Scan Models command does not remove instances from the system-identified scan instance list.

If you issue a Run command after removing a model from the user-identified scan list with the Delete Scan Models command, DFTAdvisor then has the option of including any of the instances of that model in the system-identified scan instance list.

To display the current scan model list use the Report Scan Models command.

Arguments

• model_name
  A repeatable string that specifies the model names that you want to delete from the scan model list and user-identified scan instance list. Enter the model names as they appear in the DFT library.

• -All
  A switch that specifies to delete all models from the scan model list and all instances from the user-identified scan instance list.
Delete Scan Models

Examples

The following example deletes an extra added sequential scan model; thus, the deleted model is no longer included in the scan model list:

```
set system mode dft
add scan models d_flip_flop1 d_flip_flop2
delete scan models d_flip_flop2
```

Related Commands

- Add Scan Instances
- Report Scan Models
- Add Scan Models
Delete Scan Pins

Scope: All modes

Usage

```
DELete SCan Pins chain_name... | -All
```

Description

Removes any previously assigned scan input, output, and clock names from the specified scan chains.

The Delete Scan Pins command removes user-specified names of the scan input and scan output pins of the scan chains that you previously assigned with the Add Scan Pins command. You can use the Report Scan Pins command to display all added scan input and output pin names for each scan chain.

Arguments

- `chain_name`
  
  A repeatable string that specifies the names of the scan chains from which you want DFTAdvisor to remove the associated pins.

- `-All`
  
  A switch that removes all added scan pin names from all scan chains.

Examples

The following example removes previously assigned scan chain input and output names for chain1:

```
add clocks 0 clk
set system mode dft
run
add scan pins chain1 si so
add scan pins chain2 si1 so1
delete scan pins chain1
insert test logic
```

Related Commands

Add Scan Pins  Report Scan Pins
Delete Seq\_transparent Constraints

Scope: Setup mode

Usage

DELETE seq\_transparent Constraints \{model\_name pin\_name\} \|-All

Description

Removes the pin constraints from the specified DFT library model input pins. The Delete Seq\_transparent Constraints command deletes constraints that were added to clock enable pins with the Add Seq\_transparent Constraints command. You can delete the constraints for all DFT library models or for specific pins of a model.

Arguments

- model\_name
  A string that specifies the DFT library model from whose pin\_name you want to delete the constraints.

- pin\_name
  A repeatable string specifying the clock enable pin names of the model\_name specified whose constraints you want to delete.

- -All
  A switch that specifies to delete the constraints of all DFT library model pins.

Examples

The following example adds two seq\_transparent constraints and deletes one of them:

```bash
set system mode setup
add seq\_transparent constraints c1 gdff enable
delete seq\_transparent constraints gdff enable
add seq\_transparent constraints c1 gdff en
set system mode dft
setup scan identification seq\_transparent
```
Related Commands

Add Seq_transparent Constraints  
Report Seq_transparent Constraints
Delete Sub Chains

Scope: Setup mode

Usage

DELETE SUb Chains -All | \{object_name \[subchain_name\] [-Module | -Instance]\}

Description

Removes the definition of a preexisting scan sub-chain.

The Delete Sub Chains command removes scan sub-chains previously defined with the Add Sub Chains command. The Add Sub Chains command could have been entered either directly by you, or indirectly in the scan sub-chains setup file that you might have read into DFTAdvisor. You can determine whether there are any current scan sub-chain definitions by using the Report Sub Chain command.

Arguments

- **-All**
  A switch that removes all scan sub-chains.

- **object_name**
  A string that specifies either the pathname of an instance or the name of a module. If you provide a pathname of an instance, you must also use the -Instance switch.

- **subchain_name**
  An optional string that specifies the name of a preexisting scan sub-chain that you want to remove. If you do not specify a subchain_name, DFTAdvisor removes all the subchains in the object_name.

- **-Module**
  An optional switch that specifies for DFTAdvisor to interpret the value of the object_name argument as a module name. This is the default.

- **-Instance**
  An optional switch that specifies for DFTAdvisor to interpret the value of the object_name argument as a pathname to an instance.
Examples

The following example removes a scan sub-chain:

```
add sub chains sub1 subc1 /scan_in1 /scan_out1 10 mux_scan /scan_en -module
add sub chains /I_7_22 subc2 /scan_in1 /scan_out1 6 Mux_scan /scan_en -Instance
delete sub chains /I_7_22  subc2 -instance
```

Related Commands

Add Sub Chains  Report Sub Chains
Delete Test Points

Scope: Setup mode

Usage
DELete TEst Points {-ALL | testpoint_pin_name...} [-Full | -Control | -Observe | -Lockup]

Description
Remove the test point definitions at the specified locations.

If you do not specify -Control, -Observe, or -Lockup with either the testpoint_pin_name or the -All option, DFTAdvisor uses the default -Full value. The -Full means that DFTAdvisor removes all the control and observe test points, and lockup latches at the location(s) you specify.

This command removes both user-defined and system-defined test points. You can create user-defined test points with the Add Test Points command. You can enable DFTAdvisor to automatically identify test points by using a combination of the Setup Scan Identification, Setup Test_point Identification, Setup Test_point Insertion, and Run commands.

Arguments
- **-All**
  A switch that specifies to remove the control and/or observe test points from all locations.

- **testpoint_pin_name**
  A repeatable string that specifies the locations where you want DFTAdvisor to remove the control and/or observe test points.

- **-Full**
  An optional switch that specifies to remove both the control and observe test points from the locations that you specify. This is the default.

- **-Control**
  An optional switch that specifies to remove the control test points from the locations that you specify.
Command Dictionary

Delete Test Points

- Observe

An optional switch that specifies to remove the observe test points from the locations that you specify.

- Lockup

An optional switch that specifies to remove the added lockup latch.

Examples

The following example creates the definition for three test points (one observe and two control) and then removes two of the definitions:

```
add cell models and2a -type and
add test point /I_6_16/cp control and2a in2
add test point /I_7_16/q observe out1
add test point /I_8_16/cp control and2a in3
delete test points /I_6_16/cp /I_7_16/q
```

Note

The Delete Test Point command only specifies the `testpoint_pin_names` of the test points, not the type. With this example there are two types (control and observe), which is automatically covered by the default of -Full.

Related Commands

- Add Test Points
- Insert Test Logic
- Report Test Logic
- Report Test Points
- Setup Scan Identification
- Setup Test_point Identification
- Setup Test_point Insertion
Delete Tied Signals

Scope: Setup mode

Usage

DELete TIed Signals {floating_object_name... | -All} [-Class {User | System | Full}] [-Pin]

Description

Removes the assigned (tied) value from the specified floating nets or pins.

The Delete Tied Signals command deletes the tied values that were previously assigned with the Add Tied Signals command. You can delete tied values from either user class, system class, or full classes of floating nets or pins. If you do not specify a class, the tool deletes the tied values from the user class of floating nets or pins. You can display a list of any class of tied floating nets or pins by using the Report Tied Signals command.

When you remove the effects of a tied signal, DFTAdvisor reassigns the default tied signal value to that object. The invocation default for tied objects is the unknown state (X), and you can change that default with the Setup Tied Signals command.

Arguments

• floating_object_name

A repeatable string that specifies the names of the tied floating nets or pins whose tied values you want to delete. The tool deletes all of the tied values associated with the floating nets or pins in the class of tied floating nets or pins which you specify with the -Class switch.

If you do not specify the -Pin option, the floating_object_name is assumed to be a net name. If you do specify the -Pin option, the floating_object_name is assumed to be a pin name.

• -All

A switch that specifies to delete the tied values from all tied floating nets or pins in the class of tied floating nets or pins which you identify with the -Class switch.
- **Class User | System | Full**

An optional switch and literal pair that specifies the class code of the tied floating nets or pins that you specify. The valid literal names are as follows:

- **User** — A literal that specifies that the list of tied floating nets or pins were previously added by using the Add Tied Signals command. This is the default class.

- **System** — A literal that specifies that the list of tied floating nets or pins are described in the netlist.

- **Full** — A literal that specifies that the list of tied floating nets or pins consist of both the user and system class.

- **Pin**

An optional switch that specifies that the floating_object_name argument that you provide is a floating pin name.

**Examples**

The following example deletes the tied value from the user-class tied net “vcc”; thereby leaving “vcc” as a floating net:

```plaintext
add tied signals 1 vcc vdd
delete tied signals vcc -class user
```

**Related Commands**

- Add Tied Signals
- Setup Tied Signals
- Report Tied Signals
Delete Write Controls

Scope: Setup mode

Usage

DELete WRite Controls \textit{primary\_input\_pin}... | -All

Description

Removes the RAM write control line off-state definitions from the specified primary input pins.

The Delete Write Controls command deletes write control line off-state definitions previously defined with the Add Write Controls command. You can delete the write control line definitions for specific pins or for all pins.

Arguments

\begin{itemize}
  \item \texttt{primary\_input\_pin}
    \begin{itemize}
      \item A repeatable string that specifies a list of primary input pins from which you want to delete any write control line off-state definitions.
    \end{itemize}
  \item -All
    \begin{itemize}
      \item A switch that specifies to delete the write control line off-state definitions for all primary input pins.
    \end{itemize}
\end{itemize}

Examples

The following example deletes an incorrect write control line, and adds the correct off-state to that write control line:

\begin{verbatim}
add write controls 0 w1 w2
delete write controls w1
add write controls 1 w1
set system mode dft
\end{verbatim}

Related Commands

Add Write Controls \quad Report Write Controls
Dofile

Scope: All modes

Usage

DOFile filename

Description

Executes the commands contained within the specified file.

The Dofile command sequentially executes the commands that are contained in a file which you specify. This command is especially useful when you must issue a series of commands. Rather than executing each command separately, you can place them into a file in their desired order and then execute them by using the Dofile command. You can also place comment lines in the file by starting the line with a double slash (/); DFTAdvisor handles these lines as comments and ignores them.

The Dofile command sends each command expression, in order, to the tool which in turn displays each command from the file before executing it. If DFTAdvisor encounters an error due to any command, the Dofile command stops and displays an error message. You can enable the Dofile command to continue regardless of errors by setting the Set Dofile Abort command to Off.

Arguments

• filename

A required string that specifies the name of the file that contains the commands which you want DFTAdvisor to execute.

Examples

The following example executes, all the commands from the file, command_file:

dofile command_file

The command file may contain commands like the following:

    add clocks 0 clock
    set system mode dft
    run
Dofile

Related Commands

Set Dofile Abort
Exit

Scope: All modes

Usage

EXIt [-Discard]

Description

Terminates the current DFTAdvisor session.

The Exit command terminates DFTAdvisor and returns to the operating system. You should either save the current netlist design before exiting DFTAdvisor or specify the -Discard switch to not save the netlist.

If you are operating in interactive mode (not running a dofile) and you neither saved the current netlist nor used the -Discard option, DFTAdvisor displays a warning message and you are given the opportunity to continue the session and save the netlist before exiting.

If you plan to load the scan design into FastScan or FlexTest, you may also want to save the ATPG setup to identify the scan chains before exiting.

Arguments

- -Discard

An optional switch that specifies to explicitly not save the current netlist and terminate the DFTAdvisor session.

Examples

The following example exits DFTAdvisor after performing scan chain insertion and saving the test procedure, dofile, and new netlist for the inserted scan chains:

```
add clocks 1 clk1
add clocks 0 clk0
set system mode dft
run
insert test logic
write atpg setup scan -replace
write netlist scan.edif -edif
exit
```
Related Commands

Write Atpg Setup
Write Netlist

Write Scan Identification
Extract SubcKts

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

EXTract SUbckts [subckt_name]

Description

Performs matching and conversion between the bi-directional MOS instance and the ATPG library model.

The Extract SubcKts command matches and converts matched bi-directional MOS instances to an instance that references the corresponding ATPG library model. After the extraction, if no bi-directional MOS instances exist, then you are ready to perform ATPG. Otherwise, you have to either add more SUBCKTs in the Spice SUBCKTs library or use the Add Mos Direction command to manually convert bi-directional MOS instances to uni-directional MOS instances.

Arguments

- subckt_name
  
  An optional string that specifies the name of the SUBCKT to extract. If not specified, the tool extracts SUBCKTs for all subcircuits.

Examples

The following example matches the FADD2 Spice SUBCKTs to its corresponding ATPG library model:

    extract subckts FADD2

Related Commands

Add Mos Direction Report Mos Direction
**Flatten Subckt**

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

**Usage**

```
FLAtten SUbckt subckt_name -Recursive
```

**Description**

Flattens the SUBCKT in the Spice design.

The Flatten Subckt command flattens the Spice design. Flattening enables the Extract Subckts command to perform matching of bidirectional MOS instances to ATPG library models that cross hierarchical boundaries. You can choose to flatten only the SUBCKT that crosses the hierarchical boundary.

**Arguments**

- `subckt_name`
  A required string that specifies the name of the SUBCKT you want to flatten.
- `-Recursive`
  An optional switch string that specifies that recursive flattening should occur. If omitted, the default is to flatten only one level.

**Examples**

The following example recursively flattens the FADD2 SUBCKT:

```
flatten subckt FADD2 -Recursive
```

**Related Commands**

- Extract Subckts
Help

Scope: All modes

Usage

HELP [command_name]

Description

Displays the usage syntax and system mode for the specified command.

The Help command provides quick access to either information about a specific command, to a list of commands beginning with a specific key word, or to a list of all the valid commands.

Note

The text that the Help command displays has not been fully updated for this release. For complete and up-to-date information on any command, refer to the command dictionary pages presented in this manual. For a complete list of commands supported by DFTAdvisor, refer to Table 2-1, “Command Summary” on page 2-1.

Arguments

- command_name

An optional string that either specifies the name of the command for which you want help or specifies one of the following keywords whose group of commands you want to list: ADD, DELete, SET, SETUp, or WRite. If you do not supply a command_name, the default is to display a list of all the valid command names.

Examples

The following example displays the usage and system mode for the Report Primary Inputs command:

help report primary inputs
// Report primary inputs
// usage: REPort PPrimary Inputs [-Class <User|System|Full>] [-All | pin_pathname...]  
// legal system modes: ALL
Insert Scan Chains

Scope: Dft mode

Usage

INSert SCan Chains [filename [-Fixed]] {[-NOlimit] | [-Max_length integer] | [-NUmber [integer]]} [-Clock {Nomerge | Merge}] [-Edge {Nomerge | Merge}] [-COnnect {ON | OFf | Tied}]

Description

Replaces and stitches each non-scan cell that DFTAdvisor has previously identified as a scan candidate with the corresponding scan cell.

The Insert Scan Chains command causes DFTAdvisor to replace the identified non-scan cells (scan candidates) with the corresponding scan cells, and to stitch those scan cells together into a scan chain. Scan candidates are non-scan cells that pass the system requirements so that DFTAdvisor can replace them with the corresponding scan cell. A non-scan cell can become a scan candidate by DFTAdvisor identifying it when you run the scan identification, or by you explicitly defining the cell as a scannable instance.

Note

The Insert Scan Chains command performs a subset of the functionality of the Insert Test Logic command. The recommended flow is to always use the Insert Test Logic command because it can perform all the different types of scan test logic insertions.

Arguments

- filename -Fixed

An optional string and associated optional switch that specifies the name of the ASCII file that lists the scan instances that you want DFTAdvisor to stitch together. This file can contain information regarding scan cell ordering along with which instances are to be in each scan chain.
The optional -Fixed switch specifies for DFTAdvisor to stitch the scan instances in the fixed order that is given in the filename. The default scan cell ordering is random.

The filename that you specify must list one instance per line and use the following format:

```
instance_pathname cell_id chain_id [&sub_chain_name] [+|-][lockup_latch_model]
```

instance_pathname — A string that specifies the name of the non-scan cell that you want DFTAdvisor to put in the scan chain.

cell_id — An integer that specifies the placement of the instance_pathname in relation to other instance_pathnames. DFTAdvisor places the instance having the smallest cell_id closest to the scan chain output. All instances in the same chain must have unique cell_ids.

chain_id — An integer that specifies the scan chain in which you want DFTAdvisor to place the instance_pathname. DFTAdvisor places instances with the same chain_id in the same chain.

&sub_chain_name — An optional special character and string that specifies the name of the sub-scan chain you defined with the Add Sub Chains command. You need to specify the sub-chain name when more than one sub-chain is defined for a sub-module or a hierarchical instance. No space is allowed between the ampersand (&) and the sub_chain_name argument.

{+|-}[lockup_latch_model] — An optional special character and an additional optional string that specifies the location of the lockup latch. The +|- argument specifies that a lockup latch is to be added to the scan out of the current instance_pathname. No space is allowed between the +|- and the lockup_latch_model name. Note that if you define a lockup latch in this file, you must specify every location that you want to insert lockup latches. If no lockup latches are defined in this file, DFTAdvisor uses the settings in the Set Lockup Latch command.

If you use this file, it must contain all instances you want stitched. If you do not specify a filename, DFTAdvisor stitches all non-scan cells that it has identified into a scan chain using the settings of the other Insert Scan Chains arguments.

Note

If you use this file, it must contain all instances you want stitched. If you do not specify a filename, DFTAdvisor stitches all non-scan cells that it has identified into a scan chain using the settings of the other Insert Scan Chains arguments.

The optional -Fixed switch specifies for DFTAdvisor to stitch the scan instances in the fixed order that is given in the filename. The default scan cell ordering is random.

The filename that you specify must list one instance per line and use the following format:

```
instance_pathname cell_id chain_id [&sub_chain_name] [+|-][lockup_latch_model]
```

instance_pathname — A string that specifies the name of the non-scan cell that you want DFTAdvisor to put in the scan chain.

cell_id — An integer that specifies the placement of the instance_pathname in relation to other instance_pathnames. DFTAdvisor places the instance having the smallest cell_id closest to the scan chain output. All instances in the same chain must have unique cell_ids.

chain_id — An integer that specifies the scan chain in which you want DFTAdvisor to place the instance_pathname. DFTAdvisor places instances with the same chain_id in the same chain.

&sub_chain_name — An optional special character and string that specifies the name of the sub-scan chain you defined with the Add Sub Chains command. You need to specify the sub-chain name when more than one sub-chain is defined for a sub-module or a hierarchical instance. No space is allowed between the ampersand (&) and the sub_chain_name argument.

{+|-}[lockup_latch_model] — An optional special character and an additional optional string that specifies the location of the lockup latch. The +|- argument specifies that a lockup latch is to be added to the scan out of the current instance_pathname. No space is allowed between the +|- and the lockup_latch_model name. Note that if you define a lockup latch in this file, you must specify every location that you want to insert lockup latches. If no lockup latches are defined in this file, DFTAdvisor uses the settings in the Set Lockup Latch command.
+ — Specifies that the clock used by the instance_pathname with the next lower cell_id is used. The next lower cell_id, refers to the a nonscan cell which will be connected to the scan out of the current instance_pathname.

- — Specifies that the clock used by the instance_pathname is used.

lockup_latch_model — Specifies the name of the lockup latch model to use. The specified lockup latch must be defined by the Add Cell Models command or defined in the ATPG library using the cell_type attribute. If the lockup_latch_model is not specified, the first model in the defined latch model list is used.

• -NOlimit
  An optional switch specifying that the scan chain has no limit on the number of scan cells it contains. This is the default.

• -Max_length integer
  An optional switch and integer pair that specifies the maximum number of scan cells that DFTAdvisor can stitch into a scan chain.

• -NUmber integer
  An optional switch and integer pair that specifies the exact number of scan chains that you want DFTAdvisor to insert. The default number of chains is 1.

• -Clock Nomerge | Merge
  An optional switch and literal pair that specifies whether DFTAdvisor uses different scan clocks on the same scan chain. If you turn on multiple scan enables using the Set Multiple Scan_enables command, this argument is also used to determine the number of scan enable signals. The two valid literals are as follows:

    Nomerge — A literal that specifies to not use different scan clocks on the same scan chain. This is the default.

    Merge — A literal that specifies to use different scan clocks on the same scan chain.

• -Edge Nomerge | Merge
  An optional switch and literal pair that specifies whether DFTAdvisor merges stable high chains into stable low chains. If you turn on multiple scan enables...
using the Set Multiple Scan_enables command, this argument is also used to determine the number of scan enable signals. The two valid literals are as follows:

Nomerge — A literal that specifies to not merge stable high chains into stable low chains. This is the default.

Merge — A literal that specifies to merge stable high chains into stable low chains.

• -COnnect ON | OFF | Tied

An optional switch and literal pair that specifies whether DFTAdvisor stitches the scan cells together into a scan chain. The valid literals for stitching the scan chain are as follows:

ON — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements and to stitch those scan cells together into a scan chain. This is the default.

OFF — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This options has DFTAdvisor leave the scan-specific pins floating.

Tied — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This options has DFTAdvisor tie the scan-specific pins to ground.

Examples

The following example stitches the scannable sequential instances together into scan chains with a maximum length of 10 scan cells each, then identifies 50 percent of them:

```
add clocks 0 clock
add clock groups group1 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
insert test logic -max_length 10 -clock merge
```
Insert Scan Chains

Related Commands

Setup Scan Insertion
Insert Test Logic

Scope: Dft mode

Usage

INSert TEst Logic [filename [-Fixed]] [-Scan {ON | OFF}] [-Test_point {ON | OFF}] [-Ram {ON | OFF}] [{-NOlimit} | {-Max_length integer} | {-NUmber [integer]}] [-Clock {Nomerge | Merge}] [-Edge {Nomerge | Merge}] [-COnnect {ON | OFF | Tied}] [-Output {Share | New}] [-MOdule {Norename | Rename}]

Description

Inserts the test structures that you define into the netlist to increase the design’s testability.

The purpose of DFTAdvisor is to perform tasks that modify the design to increase the fault coverage (testability) of the design. There are several ways that DFTAdvisor can aid in increasing the design’s fault coverage. The major purpose of the tool is to identify sequential cells that DFTAdvisor can replace with the corresponding scan cells.

In addition to scan cell replacement, DFTAdvisor can also perform two other tasks that can increase the design’s testability. DFTAdvisor supports the adding of test points (both system-defined and user-defined), and DFTAdvisor supports the automatic adding of test logic. For more information on scan replacement, test points, and test logic, refer to the *Scan and ATPG Process Guide*.

For information test point insertion, refer to “Automatically Choosing Control and Observe Points” in the *Scan and ATPG Process Guide*.

Arguments

- *filename* -Fixed

  An optional string and associated optional switch that specifies the name of the ASCII file that lists the scan instances that you want DFTAdvisor to stitch together. This file can contain information regarding scan cell ordering along with which instances are to be in each scan chain.
Insert Test Logic

The optional -Fixed switch specifies for DFTAdvisor to stitch the scan instances in the fixed order that is given in the filename. The default scan cell ordering is random.

The filename that you specify must list one instance per line and use the following format (all on one line):

```
instance_pathname cell_id chain_id [&sub_chain_name]
[{+|-}[lockup_latch_model]]
```

**instance_pathname** — A string that specifies the name of the non-scan cell that you want DFTAdvisor to put in the scan chain.

**cell_id** — An integer that specifies the placement of the instance_pathname in relation to other instance_pathnames. DFTAdvisor places the instance having the smallest cell_id closest to the scan chain output. All instances in the same chain must have unique cell_ids.

**chain_id** — An integer that specifies the scan chain in which you want DFTAdvisor to place the instance_pathname. DFTAdvisor places instances with the same chain_id in the same chain.

**&sub_chain_name** — An optional special character and string that specifies the name of the sub-scan chain you defined with the Add Sub Chains command. You need to specify the sub-chain name when more than one sub-chain is defined for a sub-module or a hierarchical instance. No space is allowed between the ampersand (&) and the sub_chain_name argument.

**{+|-}[lockup_latch_model]** — An optional special character and an additional optional string that specifies the location of the lockup latch. The +/- argument specifies that a lockup latch is to be added to the scan out of the current instance_pathname. No space is allowed between the +/- and the lockup_latch_model name. Note that if you define a lockup latch in this file, you must specify every location that you want to insert lockup latches. If no

---

Note: If you use this file, it must contain all instances you want stitched. If you do not specify a filename, DFTAdvisor stitches all non-scan cells that it has identified into a scan chain using the settings of the other Insert Test Logic arguments.

DFTAdvisor Reference Manual, v8.6_4
lockup latches are defined in this file, DFTAdvisor uses the settings in the `Set Lockup Latch` command.

+ — Specifies that the clock used by the `instance_pathname` with the next lower `cell_id` is used. The next lower `cell_id`, refers to the a nonscan cell which will be connected to the scan out of the current `instance_pathname`.

- — Specifies that the clock used by the `instance_pathname` is used.

`lockup_latch_model` — Specifies the name of the lockup latch model to use. The specified lockup latch must be defined by the `Add Cell Models` command or defined in the ATPG library using the `cell_type` attribute. If the `lockup_latch_model` is not specified, the first model in the defined latch model list is used.

- **-Scan ON | OFF**
  An optional switch and literal pair that specifies whether DFTAdvisor replaces the identified non-scan cells (scan candidates) with the corresponding scan cells. The valid literals are as follows:

  **ON** — A literal that enables DFTAdvisor to replace the identified non-scan cells (scan candidates) with the corresponding scan cells. This is the default.

  **OFF** — A literal that disables DFTAdvisor from replacing the identified non-scan cells (scan candidates) with the corresponding scan cells.

- **-Test_point ON | OFF**
  An optional switch and literal pair that specifies whether DFTAdvisor adds the identified test logic and test points into the design. The valid literals are as follows:

  **ON** — A literal that enables DFTAdvisor to add the identified test logic and test points into the design. This is the default.

  **OFF** — A literal that disables DFTAdvisor from adding the identified test logic and test points into the design.

- **-Ram ON | OFF**
  An optional switch and literal pair that specifies whether DFTAdvisor adds the identified test logic gates that are necessary to allow the ATPG tools access to the write control lines of the RAMs. The valid literals are as follows:
ON — A literal that enables DFTAdvisor to add the identified test logic gates for RAM write control line access. This is the default.

OFF — A literal that disables DFTAdvisor from adding the identified test logic gates for RAM write control line access.

• -NOlimit

An optional switch specifying that the scan chain has no limit on the number of scan cells it contains. This is the default.

• -Max_length  integer

An optional switch and integer pair that specifies the maximum number of scan cells that DFTAdvisor can stitch into a scan chain. DFTAdvisor evenly divides the scan cells into scan chains that are smaller than the max_length integer. Final results depend upon the number of scan candidates.

• -NUmber  integer

An optional switch and integer pair that specifies the exact number of scan chains that you want DFTAdvisor to insert. Final results depend upon the number of scan candidates. The default number of chains is 1.

• -Clock  Nomerge | Merge

An optional switch and literal pair that specifies whether DFTAdvisor uses different clocks on the same scan chain. If you turn on multiple scan enables using the Set Multiple Scan_enables command, this argument is also used to determine the number of scan enable signals. The two valid literals are as follows:

    Nomerge — A literal that specifies to not use different clocks on the same scan chain. This is the default.

    Merge — A literal that specifies to use different clocks on the same scan chain.

• -Edge  Nomerge | Merge

An optional switch and literal pair that specifies whether DFTAdvisor merges stable high chains into stable low chains. If you turn on multiple scan enables using the Set Multiple Scan_enables command, this argument is also used to determine the number of scan enable signals. The two valid literals are as follows:
Nomerge — A literal that specifies to not merge stable high chains into stable low chains. This is the default.

Merge — A literal that specifies to merge stable high chains into stable low chains.

- -COnnect ON | OFF | Tied

An optional switch and literal pair that specifies whether DFTAdvisor stitches the scan cells together into a scan chain. The valid literals for stitching the scan chain are as follows:

ON — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements and to stitch those scan cells together into a scan chain. This is the default.

OFF — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This option has DFTAdvisor leave the scan-specific pins floating.

Tied — A literal specifying for DFTAdvisor to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This option has DFTAdvisor tie the scan-specific pins to ground.

- -Output Share | New

An optional switch and literal pair that specifies how DFTAdvisor creates scan out port on submodules. The valid literals are as follows:

Share — A literal specifying that DFTAdvisor may use an existing module output port on submodules for scan out, if that port is directly connected to the scan out of a scan cell. This is the default.

New — A literal specifying that DFTAdvisor should always create a new output port for scan out.

- MOdule Norename | Rename

An optional switch and literal pair that specifies how to name the modified module. The valid literals are as follows:
Norename — A literal specifying that DFTAdvisor should use the original module name if only one type of module modification is used and that the original module is no longer used in the design. This is the default.

Rename — A literal specifying that DFTAdvisor should always rename a module if the module is modified.
Examples

The following example identifies 50 percent of the scannable sequential instances during the Run command, and then uses the Insert Test Logic command to stitch them together into scan chains with a maximum length of 10 scan cells each:

```
add clocks 0 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
insert test logic -scan on -max_length 10
```

Related Commands

- Add Scan Instances
- Add Test Points
- Set Test Logic
- Setup Scan Identification
- Setup Scan Insertion
- Setup Test_point Identification
- Setup Test_point Insertion
Mark

Tools Supported: DFTAdvisor and DFTInsight
Scope: All modes

Usage

MARk {gate_id# | pin_pathname | instance_name}... | -All | -Selected
DFTInsight Menu Path:
  Display > Mark > All | Selected

Description

Highlights the objects that you specify in the Schematic View window.

The Mark command marks objects such that the DFTInsight Schematic View
window graphically highlights them. You can mark either all the objects in the
design, individual objects that you specify, or all objects in the current selection
list. This allows you to quickly locate and identify objects in a complex schematic
view.

Additionally, many commands automatically mark key instances during
execution. Those commands that replace the display list, also automatically
unmark all system and user-marked objects.

Arguments

- **gate_id#**
  
  A repeatable integer that specifies the gate identification number of the objects
to mark. The value of the gate_id# argument is the unique identification
number that the tool automatically assigns to every gate within the design
during the model flattening process.

- **pin_pathname**
  
  A repeatable string that specifies the name of a pin whose gate you want to
mark.

- **instance_name**
  
  A repeatable string that specifies the name of the instance to mark.
• **-All**
  A switch that specifies to mark all the gates in the design.

• **-Select**
  A switch that specifies to mark all the gates in the current selection list.

### Examples

The following paragraphs provide examples of using various commands to display gates and how the mark feature is affected.

The first example displays three levels of fanout gates from the number one input of gate 51 and marks gate 51 in the display:

```
add display instances 51 -I 1 -F -Level 3
```

In this case, the marking is additive such that any instances already marked will stay marked and the key instances will be added to the marked list. If compact is ON and the 51 refers to a instance that had been compacted in the schematic created by the use of this command then the instance is not added to the marked list and you will not see the instance marked, even if you later set the compact to OFF.

The next example generates a textual display of a specific rule failure and then performs a DRC violation analysis:

```
report drc rules c4-12
// Warning: Clock /ain[0] failed rule C4 on input 3 of
  bilbo_brreg_bstage[8]_nlatch2 (14736). (C4-12)
// Source of violation: input 3 of
  bilbo_brreg_bmuxstage[8]_nlatch2 (14790)
```

```
analyze drc violation c4-12 -display
```

The Analyze Drc Violation command marks within the display the following instances which were noted by its sister command, Report Drc Rules: /ain[0], 14736, and 14790. This marking is not additive because the Analyze Drc Violation command replaces the previous schematic.

### Related Commands

- Select Object
- Unmark
- Unselect Object
- Mark
Open Schematic Viewer

Scope: All modes

Usage

OPEN SChematic VViewer

Description

Invokes the optional schematic viewing application, DFTInsight.

The Open Schematic Viewer command opens a DFTInsight schematic display window. If you issue the Analyze Drc or the Add Display Instances command prior to the Open Schematic Viewer command, DFTInsight displays that netlist upon invocation. Otherwise, the schematic display window is initially empty. The DFTInsight schematic viewer handles two types of interrupts as follows:

- Terminating a Large Schematic Generation: When DFTInsight generates a large schematic, it may take several minutes. You can terminate a lengthy generation by entering Control-C in the DFTInsight window. This causes the display to revert back to the previously viewed schematic. If you enter Control-C multiple times, the first Control-C terminates the schematic generation as described and all others are trapped and discarded.

- Terminating a Dynamic View or Select Area: When using the mouse to perform a view area or select area by using the press-drag-release or click-move-click methods, you can terminate the dynamic by pressing the Escape key. This leaves the schematic in its state prior to initiating the view or select area.

Examples

The following example invokes the schematic viewer, creates and displays a netlist:

```
open schematic viewer
analyze drc violation c2-1
```

Related Commands

Close Schematic Viewer Set Schematic Display
## Read Subckts Library

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

### Usage

`READ Subckts Library filename`

### Description

Reads the specified Spice SUBCKT library.

The Read Subckts Library command specifies the Spice netlist file to read. This file contains various SUBCKTs. Each SUBCKT should have one corresponding ATPG library model. The relationship between the SUBCKT and the ATPG library model is established by the use of the same name. If a SUBCKT exists that does not have a corresponding ATPG library model, it is discarded.

### Arguments

- `filename`

  A required path and filename of the SPICE netlist containing various SUBCKTs.

### Examples

The following example reads the Spice SUBCKT library specified:

```
read subckts library C51.sp
```

### Related Commands

- Extract Subckts
**Redo Display**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must have the optional DFTInsight application invoked, you must have issued an Undo command, and not have added or deleted any instances to or from the schematic since the undo.

**Usage**

REDo DIsplay [level]

DFTInsight Menu Path:
   
   Display > Redo > One Level | N Levels

**Description**

Nullifies the schematic view effects of an Undo command.

The Redo Display command nullifies the number of Undo Display commands that you specify. This restores the DFTInsight schematic view prior to the last nullified Undo Display command. The maximum undo history level is 19.

**Arguments**

- **level**

   An optional positive integer that specifies the number of Undo Display commands that you want to nullify. The integer value cannot exceed the number of qualified Undo Display commands. A qualified Undo Display command is one that has not been followed by any other command that added or deleted any instances to or from the netlist. The default level is 1.

**Examples**

The following series of examples shows how to display several different schematics, each overwriting the last, and then how to undo and redo the schematic displays.
The first example invokes DFTInsight, then displays four custom gate paths by specifying the first and last gate identification numbers for each path:

```
open schematic viewer
add display path 23 51
add display path 51 88
add display path 51 65
add display path 65 102
```

The DFTInsight schematic view now displays all the gates between gate 65 and gate 102.

The next example undoes the last three schematic displays and restores (reverts back to) the schematic view display of all the gates between gate 23 and gate 51:

```
undo display 3
```

The final example redoes (or nullifies) the last two undo operations and restores the schematic view display of all the gates between gate 51 and gate 65:

```
redo display 2
```

**Related Commands**

- Open Schematic Viewer
- Undo Display
Report Atpg Constraints

Scope: Bist mode

Usage
REPort ATpg Constraints

Description
Displays all the current ATPG state restrictions and the instance pins on which they reside.

The Report Atpg Constraints command displays the pins and their state restrictions defined using the Add Atpg Constraints command. The tool uses the state restrictions (constraints) during the ATPG process.

Examples
The following example creates two ATPG pin constraints and then displays the information on those definitions:

```
add atpg constraints 0 /i$135/q
add atpg constraints 1 /i$141/q
report atpg constraints
0 /I$135/Q (23)
1 /I$135/Q (24)
```

Related Commands
Add Atpg Constraints Delete Atpg Constraints
Report Buffer Insertion

Scope: All modes

Usage

REPort BUffer Insertion

Description

Displays a list of all the different scan test pins and the corresponding fanout limit.

The Report Buffer Insertion command shows either the default setting (infinity) for each type of scan test pin or the new limit you set with the Add Buffer Insertion command.

Examples

The following example changes the fanout limit with the Add Buffer Insertion command, then reports the new setting and all the remaining settings are left at the default (infinity):

```bash
add buffer insertion 7 sen -model buf1a
report buffer insertion
scan_enable          7    buf1a
scan_clock           <infinity>
test_enable          <infinity>
test_clock           <infinity>
scan_master_clock    <infinity>
scan_slave_clock     <infinity>
hold_enable          <infinity>
```

Related Commands

Add Buffer Insertion                     Delete Buffer Insertion
Report Cell Models

Scope: All modes

Usage

REPort CEll Models [-All | {-Type cell_model_type}]

Description

Displays a list of either all cell models or the DFT library models associated with the specified cell type.

The Report Cell Models command displays the cell models that you either added with the Add Cell Models command, or described in the DFT library with the cell_type attribute.

Arguments

• -All
  An optional switch that specifies to display all cell model definitions that you added with the Add Cell Models command. This is the default.

• -Type cell_model_type
  An optional switch and literal pair that specifies to display a listing of all the cell models of a particular type. The valid cell_model_types are as follows (with the minimum typing characters shown in uppercase):

  INV — A literal that specifies a one-input inverter gate.

  And — A literal that specifies a two-input AND gate.

  Buf — A literal that specifies a one-input buffer gate.

  OR — A literal that specifies a two-input OR gate.

  NAnd — A literal that specifies a two-input NAND gate.

  NOR — A literal that specifies a two-input NOR gate.

  Xor — A literal that specifies a exclusive OR gate.

  INBuf — A literal that specifies a primary input buffer gate that DFTAdvisor inserted whenever the tool added new input pins (such as the scan input or scan enable pins).
OUtbuf — A literal that specifies a primary output buffer gate that DFTAdvisor inserted whenever the tool added new output pins (such as the scan output pin).

Mux — A literal that specifies a 2-1 multiplexer.

Scancell — A literal that specifies a cell with four input pins (clock, data, scan in, and scan enable), clocked scan cell with four inputs (clock, data, scan clock, and scan enable), or LSSD scan cell with five inputs (clock, data, scan in, master clock, and slave clock).

DFf — A literal that specifies a D flip-flop with two input pins (clock and data).

DLat — A literal that specifies a D latch with two input pins (enable and data).

Examples

The following example displays a list of all added cell models:

```
add clocks 0 clk
set test Logic -set on -re on -clock on
set system mode dft
report dft check
add cell models and2 -type and
add cell models or2 -type or
add cell models mux21h -type mux s a b
add cell models nor2 -type nor
report cell models
insert test logic
```

Related Commands

Add Cell Models
Delete Cell Models
Set Test Logic
Report Clock Groups
Scope: Dft mode

Usage
REPort CLock Groups

Description
Displays a list of all clock group definitions.
The Report Clock Groups command displays a list of all clock groups added with the Add Clock Groups command.

Examples
The following example displays a list of clock groups after they have been added to the clock list:

```
add clocks 1 clk1 clk2
add clocks 0 clr1 clr2 pre1 pre2
set system mode dft
...
add clock groups group1 clk1 clr1 pre1
add clock groups group2 clk2 clr2 pre2
report clock groups
  group2: clk2 clr2 pre2
  group1: clk1 clr1 pre1
```

Related Commands
Add Clock Groups Delete Clock Groups
Report Clocks

Scope: All modes

Usage

REPort CLocks

Description

Displays a list of all clock definitions.

The Report Clocks command displays a list of all clocks added with the Add Clocks command.

Examples

The following example displays a list of clocks after they have been added to the clock list:

```
add clocks 1 clk1
add clocks 0 clk0
report clocks
    clk1, off_state 1
    clk0, off_state 0
```

Related Commands

Add Clocks  Delete Clocks
Report Control Signals

Scope: Dft mode

Usage


Description

Displays the rules checking results for the specified control signals.

The Report Control Signals command displays the rules checking results for control signals consisting of clocks added with the Add Clocks command and pins identified for gating scannable memory elements with test logic.

Arguments

- **-All**
  An optional switch specifying to report on all control signals. This is the default.

- **pin_pathname**
  An optional repeatable string specifying the pathnames of control signal signals for which you want a report.

- **-Clock**
  An optional switch specifying to report on all clock control signals.

- **-Set**
  An optional switch specifying to report on all set control signals.

- **-Reset**
  An optional switch specifying to report on all reset control signals.

- **-Write**
  An optional switch specifying to report on all write control signals.
• **-Read**
  An optional switch specifying to report on all read control signals.

• **-Tristate_enable**
  An optional switch specifying to report on all tristate enable signals.

• **-Verbose**
  An optional switch specifying to include all memory elements associated with each control signal in the report.

• **-NOSTABLE_High**
  An optional switch specifying not to report on any stable-high memory elements.

• **-NOSTABLE_Low**
  An optional switch specifying not to report on any stable-low memory elements.

**Related Commands**

- Add Clocks
- Delete Clocks
- Report Dft Check
Report Dft Check

Scope: Dft mode

Usage

REPort Dft Check [-All | instance_pathname] {[-FIlename filename] [-REplace]} [-FUll | -Scannable | -Nonscannable | {-Defined {Scan | Nonscan} | -Identified | -Unidentified | {-RUle {S1 | S2 | S3}} | -Tristate | -RAm]

Description

Generates the scannability check results for non-scan instances.

The Report Dft Check command generates scannability check information for all non-scan instances in the design or within a specific hierarchical instance. The displayed or written report includes six columns of information as described here:

- The first column displays whether the DFT Rules Checker identified the non-scan instance as scannable or non-scannable.
- The second column displays whether the non-scan instance is unidentified, identified by the scan identification process, or defined as non-scannable or scannable with the Add Nonscan Instances and Add Scan Instances commands, respectively.
- The third column displays the clock that is associated with the non-scan instance, or displays “Test-Logic” to specify that test logic will be added to make the instance scannable. If the non-scan instance is identified as non-scannable, the third column displays the design rule ID number, where the non-scan instance failed the clock rules. The gate index number of the non-scannable instance is also shown, as well as the set, reset or clock primary inputs to the memory element that failed the rules checking.
- The fourth column displays the instance name of the non-scan instance.
- The fifth column displays the library model name associated with the instance.
- The sixth column displays “Stable-high” if the clock inputs of the non-scan instances are at a one-state, with respect to the clock primary inputs.
An example of the output of this command, along with additional information, is covered in “Reporting Scannability Information” in the *Scan and ATPG Process Guide*.

**Arguments**

- **-All**
  
  An optional switch that specifies to display all non-scan instances for the entire design. This is the default.

- **instance_pathname**
  
  An optional string that specifies to report scannability information for the specified instance. If the instance pathname specifies a hierarchical instance, DFTAdvisor generates scannability information for all instances within that hierarchical block. If the instance pathname specifies a particular sequential instance, DFTAdvisor generates scannability information for only that instance.

- **-Filename filename**
  
  An optional switch and string pair that specifies the name of the file to which DFTAdvisor writes the scannability check information.

- **-REplace**
  
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.

- **-Full**
  
  An optional switch that specifies to display the full scannability check information for all non-scan instances. This is the default.

- **-Scannable**
  
  An optional switch that specifies to display the non-scan instances that DFTAdvisor has identified during the rules checking process to be scannable.

- **-Nonscannable**
  
  An optional switch that specifies to display the non-scan instances that DFTAdvisor has identified during the rules checking process to be non-scannable.
-Defined Scan | Nonscan

An optional switch and literal pair that specifies whether to display user-defined scan or non-scan instances. The valid literals are as follows:

  Scan — A literal that specifies to display scan instances defined with the Add Scan Instances command.

  Nonscan — A literal that specifies to display non-scan instances defined with the Add Nonscan Instances command.

-Identified

An optional switch that specifies to display the non-scan instances that DFTAdvisor has identified with the scan identification process.

-Unidentified

An optional switch that specifies to display non-scan instances that remain to be identified by the scan identification process.

-Rule S1 | S2 | S3

An optional switch and literal specifying that nonscannable cells violating the specified rule (S1, S2, or S3) are reported. For more information on S1, S2, and S3 rule violations, refer to “Scannability Rules” in the Design-for-Test Common Resources Manual.

-Tristate

An optional switch that specifies to display the enable lines of tri-state gates that are connected from the outputs of memory elements.

-RAm

An optional switch that specifies to display the RAM gates that DFTAdvisor has identified to be controllable through test logic insertion.

Examples

The following example displays the scannability check for all non-scan instances in the design:

```plaintext
add clocks 1 clk1
add clocks 0 clk0
set system mode dft
report dft check
```
<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
<th>Clock</th>
<th>Instance</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCANNABLE</td>
<td>DEFINED-NONSCAN</td>
<td>/CLK1</td>
<td>/U1</td>
<td>FD1</td>
</tr>
<tr>
<td>SCANNABLE</td>
<td>IDENTIFIED</td>
<td>/CLK1</td>
<td>/U2</td>
<td>FD1</td>
</tr>
<tr>
<td>SCANNABLE</td>
<td>UNIDENTIFIED</td>
<td>/CLK1</td>
<td>/U3</td>
<td>FD1</td>
</tr>
<tr>
<td>SCANNABLE</td>
<td>DEFINED-SCAN</td>
<td>/CLK1</td>
<td>/U4</td>
<td>FD1</td>
</tr>
<tr>
<td>SCANNABLE</td>
<td>UNIDENTIFIED</td>
<td>/CLK2</td>
<td>/U5</td>
<td>FD1</td>
</tr>
<tr>
<td>SCANNABLE</td>
<td>IDENTIFIED</td>
<td>/CLK2</td>
<td>/U6</td>
<td>FD1</td>
</tr>
<tr>
<td>NON-SCANNABLE</td>
<td>UNIDENTIFIED</td>
<td>S1</td>
<td>/U7</td>
<td>FD2  (34)</td>
</tr>
</tbody>
</table>

Clock #1: /CLK3 (11)

Number of non-scannable instances fails on S1 rule = 1
Number of instances found = 1
Number of instances reported = 1

**Related Commands**

- Report Drc Rules
Report Display Instances

Tools Supported: DFTAdvisor and DFTInsight
Scope: Bist and Dft modes
Prerequisites: You must have the optional DFTInsight application invoked.

Usage

REPort DIspaly Instances {{gate_id# | instance_name}... | -All} [-Full]
DFTInsight Menu Path:
Report (Instance popup menu)

Description

Displays a textual report of the netlist information for either the gates or instances that you specify or for all the gates in the current schematic view display.

The Report Display Instance command causes DFTInsight to transcript the information that you request to the message area of the DFTInsight session (which is below the schematic view window). By default, the Report Display Instances command displays the following:

- Instance pathname
- Gate identification number
- Primitive type

If you do not have access to the optional DFTInsight application, you can display the same information within DFTAdvisor by using the Report Gates command.

Arguments

- gate_id#

A repeatable integer that specifies the gates whose netlist information you want DFTInsight to transcript. The value of the gate_id# argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.
• **instance_name**
  A repeatable string that specifies the name of a top-level instance within the design whose netlist information you want DFTInsight to transcript. DFTInsight generates a report on the gate associated with that `instance_name`.

• **-All**
  A switch that specifies to generate a report on all the objects in the current display netlist (including the compacted gates).

• **-Full**
  An optional switch that specifies to also include the pin information in the report. The pin information includes the following:

  - Pin name
  - Pin type (input or output)
  - Simulated pin data (if appropriate)
  - Gates to which that pin is connected

**Examples**

The following example invokes the optional schematic viewing application, displays the gates associated with a specific design rule violation, and then sets the gate reporting to display the simulation data associated with the error:

```plaintext
open schematic viewer
analyze drc violation c2-1
set gate report error_pattern
```

The next two commands show the differences between the high-level and the detailed reports:

```plaintext
report display instances l_2_16
  // /I_2_16/DFF0 (58) DFF

report display instances l_2_16 -full
  // /I_2_16/DFF0 (58) DFF
  //    SET  I (0) 51-
  //    RESET I (1) 47-
  //    CLK  I (X) 17-
  //    DATA I (X) 20-
  //    “OUT” O (X) 15- 16-
```
Related Commands

Add Display Instances  Report Gates
Analyze Drc Violation  Set Gate Report
Open Schematic Viewer
**Report Drc Rules**

Scope: All modes

**Usage**

REPort DRc Rules [*rule_id-occurrence#*] [-Summary] [-Verbose]

**Description**

Displays either a summary of all the Design Rule Check (DRC) violations or the data for a specific violation.

The Report Drc Rules command displays the following information for a specific violation:

- Rule identification number
- Current number of rule failures
- Violation handling
- ATPG analysis flag (if used)
- Rule verbosity flag (if used).

You can use the Set Drc Handling command to change the handling of the C (clock), A (RAM), D (data), P (procedure), T (trace), and E (extra) rules. For more information on the design rules, refer to the “Design Rules Checking” section in the Design-for-Test Common Resources Manual.

**Arguments**

- *rule_id-occurrence#

A literal that specifies the identification of the exact design rule violation (including the occurrence) for which you want to display information. The argument must include the design rules violation ID (*rule_id*), the specific occurrence number of that violation, and the hyphen between them. For example, you can analyze the second occurrence of the C3 rule by specifying C3-2. The tool assigns the occurrences of the rules violations as it encounters them; you cannot change either the rule identification number or the ordering of the specific violations.
The design rule violations and their identification literals divide into the following six groups: RAM, Clock, Data, Extra, Scannability, and Trace rules violation IDs.

The following lists the RAM rules violation IDs. For a complete description of these violations refer to the “RAM Rules” section in the Design-for-Test Common Resources Manual.

A1 — When all write control lines are at their off-state, all write, set, and reset inputs of RAMS must be at their inactive state.

A2 — A defined scan clock must not propagate to a RAM gate, except for its read lines.

A3 — A write or read control line must not propagate to an address line of a RAM gate.

A4 — A write or read control line must not propagate to a data line of a RAM gate.

A5 — A RAM gate must not propagate to another RAM gate.

A6 — All the write inputs of all RAMs and all read inputs of all data_hold RAMs must be at their off-state during all test procedures, except test_setup.

A7 — When all read control lines are at their off-state, all read inputs of RAMs with the read_off attribute set to hold must be at their inactive state.

The following lists the Clock rules violation IDs. For a complete description of these violations refer to the “Clock Rules” section in the Design-for-Test Common Resources Manual.

C1 — The netlist contains the unstable sequential element in addition to the backtrace cone for each of its clock inputs. The pin data shows the value that the tool simulates when all the clocks are at their off-states and when the tool sets all the pin constraints to their constrained values.

C2 — The netlist contains the failing clock pin and the gates in the path from it to the nearest sequential element (or primary input if there is no sequential element in the path.) The pin data shows the value that the tool simulates when the failing clock is set to X, all other clocks are at their off-states, and when the tool sets all pin constrains to their constrained values.
C3 | C4 — The netlist contains all gates between the source cell and the failing cell, the failing clock and the failing cell, and the failing clock and the source cell. The pin data shows the clock cone data for the failing clock.

C5/C6 — The netlist contains all gates between the failing clock and the failing cell. The pin data shows the clock cone data for the failing clock.

C7 — The netlist contains all the gates in the backtrace cone of the bad clock input of the failing cell. The pin data shows the constrained values.

C8 | C9 — The netlist contains all the gates in the backtrace cone of the failing primary output. The pin data shows the clock cone data for the failing clock.

The following lists the Data rules violation IDs. For a complete description of these violations refer to the “Scan Cell Data Rules” section in the Design-for-Test Common Resources Manual.

D1 — The netlist contains all the gates in the backtrace cone of the clock inputs of the disturbed scan cell. The pin data shows the pattern values the tool simulated when it encountered the error.

D2 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulated for all time periods of the shift procedure.

D3 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the master_observe procedure.

D4 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the skew_load procedure.

D5 — The netlist contains the disturbed gate, and there is no pin data.

D6 | D7 | D8 — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the value that the tool simulates when all clocks are at their off-states.

D9 — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the pattern value the tool simulated when it encountered the error.
The following lists the Extra rules violation IDs. For a complete description of these violations refer to the “Extra Rules” section in the Design-for-Test Common Resources Manual.

E2 — There must be no inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell.

E3 — There must be no inversion between MASTER and SLAVE for any scan cell.

E4 — Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures.

E5 — When constrained pins are at their constrained states, and PIs and scan cells are at their specified binary states, X states must not be capable of propagating to an observable point.

E6 — When constrained pins are placed at their constrained states, the inputs of a gate must not have sensitizable connectivity to more than one memory element of a scan cell.

E7 — External bidirectional drivers must be at the high-impedance (Z) state during the application of the test procedure.

E9 — The drivers of wire gates must not be capable of driving opposing binary values.

E10 — Performs bus contention mutual-exclusivity checking. Similar to E4, but does not check for this condition during test procedures.

E11 — The ability of a bus gate to attain a Z state.

E12 — The test procedures cannot violate any ATPG constraints.

E13 — Satisfy both ATPG constraints and bus contention prevention (for buses that fail rule E10)

The following lists the Scannability rules violation IDs. For a complete description of these violations refer to the “Scannability Rules” section in the Design-for-Test Common Resources Manual:

S1 — Checks all the clock inputs (including sets and resets) of each nonscan memory element to ensure that these inputs can be turned off.
S2 — Checks all clock inputs (not including sets and resets) of each nonscan memory element to see whether they can capture data.

S3 — Checks for mux-DFF style scan to see if defined clocks can be used as shift clocks.

The following lists the Trace rules violation IDs. For a complete description of these violations refer to the “Scan Chain Trace Rules” section in the Design-for-Test Common Resources Manual.

T2 — The netlist contains the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T3 — The netlist contains all the gates in the backtrace cone of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T4 — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T5 | T6 — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T7 — The netlist contains all the gates in the path between the two failing latches. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T11 — A clock input of the memory element closest to the scan chain input must not be turned on during the shift procedure prior to the time of the force_sci statement.

T16 — When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink gate (PGS) in the long path must be at the non-controlling value of the PGS gate.

T17 — Reconvergent pulse generator sink gates cannot be connected to any of the following: primary outputs, non-clock inputs of the scan memory elements, ROM gates, non-write inputs of RAMs and transparent latches.
• **Summary**
  A switch that displays the following for each user-controllable rule (default):
  - Rule identification number
  - Number of failures of each rule
  - Current handling status of that rule

• **Verbose**
  A switch that displays the following for each user-controllable rule:
  - Rule identification number
  - Number of failures of each rule
  - Current handling status of that rule
  - Brief description of that rule.

**Examples**

The following example changes the severity of the data rule 7 (D7) from a warning to an error and also specifies execution of a full test generation analysis when performing the rules checking for the clock (C) rules. Next, the example generates a display of a specific rule failure:

```plaintext
set drc handling d7 error atpg_analysis
set system mode dft
//-----------------------------------------------------------
//Begin scan chain identification process, memory elements=8.
//-----------------------------------------------------------
// Reading group test procedure file /user/design/tpf.
// Simulating load/unload procedure in g1 test procedure file.
// Chain = c1 successfully traced with scan_cells = 8.
// Error: Flipflop /FF1 (103) has clock port set to stable high.(D7-1)
// Error: Rules checking unsuccessfule, cannot exit SETUP mode.
report drc rules d7-1
//Error: Flipflop /I$3 (16) has clock port set to stable high (D7-1)
```

**Related Commands**

- Set Drc Handling
Report Environment

Scope: All modes

Usage

REPort ENvironment

Description

Displays the current values of all the “set” commands and the default names of the scan type pins.

When you first invoke DFTAdvisor, executing the Report Environment command shows all of the default values of the “set” commands.

Examples

The following example reports the DFTAdvisor invocation defaults:

```
report environment
Top Module = /designs/dft/test_design
Gate Level = design
Gate Report = normal
Net Resolution = wire
System Mode = setup
Tied Signal = x
Dofile Abort = on
Trace Report = off
Scan type = mux_scan
Identification Type = sequential:on scan_sequential:off
  partition_scan:off full_scan:off test_point:off
Identification Model = clock:original disturb:on
Scan Identification = automatic Internal Full backtrack=30
  cycle=16 time=100 control_coverage = 100
  observe_coverage = 100
min_detection = 1
Fault Sampling = 100%
Scan-in Naming = prefix:scan_in initial:1 modifier:1 suffix:
Scan-out Naming = prefix:scan_out initial:1 modifier:1
  suffix:
Test Enable Name = test_en active = high
Test Clock Name = test_clk
Scan Enable Name = scan_en
```
Scan Clock Name = scan_clk
Scan Master Clock Name = scan_mclk
Scan Slave Clock Name = scan_sclk
Hold Enable Name = hold_en
Control Input Name = test_cntl
Observe Output Name = test_obs
Test Logic = set:off reset:off clock:off tristate:off ram:off
Screen Display = on
LockupLatch = off nolast

**Related Commands**

Any of the “Set” commands.
Report Faults
Scope: Bist mode

Usage
REPort FAults [-Class class_type] [-Stuck_at {01 | 0 | 1}] [-All | object_pathname...] [-Hierarchy integer [-Min_count integer]] [-Noeq]

Description
Displays fault information from the current fault list.
The Report Faults command displays faults from the fault list added using the Add Faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at faults that occur on a specific object in a specific class. If you do not specify any arguments, Report Faults displays information on all the known faults.
The Report Faults command displays the following three columns of information for each fault:

- fault value - The fault value may be either 0 (for stuck-at-0) or 1 (for stuck-at-1).
- fault code - A code name indicating the lowest level fault class assigned to the fault.
- fault site - The pin pathname of the fault site.
You can use the -Hierarchy option to display a hierarchal summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option which specifies the minimum number of faults that must be in a hierarchal level before displaying.

Arguments
- -Class class_type
  An optional switch and literal pair that specifies the class of faults that you want to display. The class_type argument can be either a fault class code or a fault class name. If you do not specify a class_type, the command displays all
fault classes. Table 2-2 lists the valid fault class codes and their associated fault class names; use either the code or the name when specifying the class_type argument:

Table 2-2. Fault Class Codes and Names

<table>
<thead>
<tr>
<th>Fault Class Codes</th>
<th>Fault Class Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU</td>
<td>Full</td>
</tr>
<tr>
<td>TE</td>
<td>TEstable</td>
</tr>
<tr>
<td>DT</td>
<td>DETEcted</td>
</tr>
<tr>
<td>DS</td>
<td>DET_Simulation</td>
</tr>
<tr>
<td>DI</td>
<td>DET_Implication</td>
</tr>
<tr>
<td>DR</td>
<td>DET_Robust (Path Delay Testing Only)</td>
</tr>
<tr>
<td>PD</td>
<td>POSDET</td>
</tr>
<tr>
<td>PU</td>
<td>POSDET_Untestable</td>
</tr>
<tr>
<td>PT</td>
<td>POSDET_Testable</td>
</tr>
<tr>
<td>OS</td>
<td>OSCIIllatory (FlexTest Only)</td>
</tr>
<tr>
<td>OU</td>
<td>OSC_Untestable</td>
</tr>
<tr>
<td>OT</td>
<td>OSC_Testable</td>
</tr>
<tr>
<td>HY</td>
<td>HYPErtrophic (FlexTest Only)</td>
</tr>
<tr>
<td>HU</td>
<td>HYP_Untestable</td>
</tr>
<tr>
<td>HT</td>
<td>HYP_Testable</td>
</tr>
<tr>
<td>UI</td>
<td>Uninitializable (FlexTest Only)</td>
</tr>
<tr>
<td>AU</td>
<td>Atpg_untestable</td>
</tr>
<tr>
<td>UD</td>
<td>UNDetected</td>
</tr>
<tr>
<td>UC</td>
<td>UNControlled</td>
</tr>
<tr>
<td>UO</td>
<td>UNObserved</td>
</tr>
<tr>
<td>UT</td>
<td>UNTTestable</td>
</tr>
</tbody>
</table>
• \textbf{-Stuck\_at 01 | 0 | 1} \\
An optional switch and literal pair that specifies the stuck-at faults that you want to display. The stuck-at literal choices are as follows:

- \textbf{01} — A literal that displays both the “stuck-at-0” and “stuck-at-1” faults. This is the default.
- \textbf{0} — A literal that displays only the “stuck-at-0” faults.
- \textbf{1} — A literal that displays only the “stuck-at-1” faults.

• \textbf{-All} \\
An optional switch that displays all of the faults on all model, netlist primitive, and top module pins. This is the default.

• \textbf{object\_pathname} \\
An optional repeatable string that specifies a list of pins or instances whose faults you want to display.

• \textbf{-Hierarchy integer} \\
An optional switch and integer pair that specifies the maximum fault class hierarchy level for which you want to display a hierarchal summary of the faults.

• \textbf{-Min\_count integer} \\
An optional switch and integer pair that you can use with the -Hierarchy option and that specifies the minimum number of faults that must be in a hierarchal level to display the hierarchal summary. The default is 1.
• -Noeq
   An optional switch that displays the fault class of equivalent faults. When you do not specify this switch, the tool displays an “EQ” as the fault class for any equivalent faults.

Examples
The following example displays all faults that have been added to the circuit before performing a fault simulation and signature calculation run:

   set system mode bist
   add faults -all
   report faults -all
   run

Related Commands
Add Faults          Delete Faults
Report Feedback Paths

Tools Supported: DFTAdvisor and DFTInsight

Scope: Bist and Dft mode

Prerequisites: You can use this command only after the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode.

Usage

REPort FEeeback Paths

DFTInsight Menu Path:
  Display > Additions: Loops

Description

Displays a textual report of the currently identified feedback paths.

The Report Feedback Paths command lists the identification numbers of any feedback paths that the tool identified during the last circuit learning process. These feedback paths include, by default, any duplicated gates. You can suppress duplicated gates by using the Set Loop Duplication command prior to initiating the circuit learning process.

As stated earlier, the Report Feedback Paths command displays all the feedback path identification numbers. You can use these identification numbers with the Add Display Loop command to schematically display specific feedback paths. When you issue the Add Display Loop command for specific feedback paths, DFTInsight transcripts the same information as the Report Feedback Paths command but, only for the paths that you specified.

Examples

The following example invokes the optional schematic viewing application, leaves the Setup mode (which, among other things, flattens the simulation model and performs the learning process), displays the identification numbers of any learned feedback paths, and then schematically displays one of the feedback paths:

  open schematic viewer
  set system mode dft
report feedback paths
Loop#=0, feedback_buffer=26, #gates_in_network=5
   INV   /I_956__I_582/ (51)
   PBUS  /I_956__I_582/N1/ (96)
   ZVAL  /I_956__I_582/N1/ (101)
   INV   /I_956__I_582/ (106)
   TIEX  /I_956__I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
   INV   /I_962__I_582/ (52)
   PBUS  /I_962__I_582/N1/ (95)
   ZVAL  /I_962__I_582/N1/ (100)
   INV   /I_962__I_582/ (105)
   TIEX  /I_962__I_582/ (27)

add display loop 1

Related Commands
Add Display Loop          Set Loop Duplication
Report Flatten Rules

Scope: All modes

Usage

REPort FLatten Rules [rule_id [{ocurrence_id | -Verbose}]]

Description
Displays either a summary of all the flattening rule violations or the data for a specific violation.

The Report Flatten Rules command displays the following information for a specific violation:

- Rule identification number
- Current number of rule failures
- Violation handling

You can use the Set Flatten Handling command to change the handling of the net, pin, and gate rules.

Arguments

- rule_id

A literal that specifies the flattening rule violation for which you want to display information. The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

Following are the net rules:

- **FN1** — A module net is floating. The default upon invocation is warning.
- **FN2** — A module net has driver and constant value property. The default upon invocation is warning and its property is not used.
- **FN3** — An instance net is floating. The default upon invocation is warning.
- **FN4** — An instance net is not used. The default upon invocation is warning.
FN5 — A multiple driven wired net. The default upon invocation is warning.

FN6 — A bus net attribute cannot be used. The default upon invocation is warning.

FN7 — Two connected nets have inconsistent net attributes. The default upon invocation is warning and both attributes are not used.

FN8 — Parallel wired behavior. The default upon invocation is warning.

FN9 — The bus net has multiple different bus keepers. The default upon invocation is warning and their effects are additive.

Following are the pin rules:

FP1 — The circuit has no primary inputs. The default upon invocation is warning

FP2 — The circuit has no primary outputs. The default upon invocation is warning.

FP3 — The primary input drives logic gates and switch gates. The default upon invocation is warning.

FP4 — A pin is moved. The default upon invocation is warning.

FP5 — A pin was deleted by merging. The default upon invocation is warning.

FP6 — Merged wired in/out pins. The default upon invocation is warning

FP7 — Merged wired input and output pins. The default upon invocation is warning

FP8 — A module boundary pin has no name. The default upon invocation is warning

FP9 — An in/out pin is used as output only. The default upon invocation is ignored

FP10 — An output pin is used as in/out pin. The default upon invocation is ignored

FP11 — An input pin is used as in/out pin. The default upon invocation is ignored
FP12 — An output pin has no fan-out. The default upon invocation is ignored

FP13 — An input pin is floating. The default upon invocation is warning

Following are the gate rules:

FG1 — The defining model of an instance does not exist. The default upon invocation is error. If it is not an error condition, this instance is treated as an undefined primitive.

FG2 — The feedback gate is not in feedback loop. The default upon invocation is error.

FG3 — The bus keeper has no functional impact. The default upon invocation is warning

FG4 — The RAM/ROM read attribute not supported. The default upon invocation is warning

FG5 — The RAM attribute not supported. The default upon invocation is warning

FG6 — The RAM type not supported. The default upon invocation is error

FG7 — The netlist module has a primitive not supported. The default upon invocation is error. If non-error is chosen, this primitive is treated as undefined.

FG8 — The library model has a primitive not supported. The default upon invocation is error. If non-error is chosen, this primitive is treated as undefined.

• occurrence_id

A literal that specifies the identification of the exact flattening rule violation (the occurrence) for which you want to display information. For example, you can analyze the second occurrence of the FG4 rule by specifying the rule_id and the occurrence_id, FG4 2. The tool assigns the occurrences of the rules violations as it encounters them; you cannot change either the rule identification number or the ordering of the specific violations.
- **-Verbose**
  
  A switch that displays the following for each flattening rule:
  
  - Rule identification number
  - Number of failures of each rule
  - Current handling status of that rule
  - Brief description of that rule.

**Example**

The following example shows the summary information of the FG3 rule:

```
report flatten rules fg3
// FG3: fails=2 handling=warning/noverbose
```

**Related Commands**

- Set Flatten Handling
Report Gates

Scope: All modes

Prerequisites: Although you can use this command in Setup or Dft mode, you can use it in the Setup mode only after the netlist has been flattened. This happens when you first attempt to exit Setup mode. Next time you return to Setup mode you can use the command.

Usage

REPORT GATES {gate_id# | pin_pathname | instance_name}... | {-Type gate_type}...

Description

Displays the netlist information for the specified gates.

The Report Gates command displays the netlist information for either the design-level or primitive-level gates that you specify. You specify the gate by its gate index number, a pathname of a pin connected to a gate, an instance name (design level only), or a gate type.

You can specify a design cell by a pathname of a pin connected to the design cell.

If you use a gate index number or gate type, the command always reports the primitive-level.

The format for the design level is:

```
instance_name cell_type
  input_pin_name    I    (data)    pin_pathname...
  ...               
  output_pin_name   0    (data)    pin_pathname...
  ...
```

The format for the primitive level is:

```
instance_name (gate_ID#) gate_type
  input_pin_name    I    (data)    gate_ID#-pin_pathname...
  ...
  output_pin_name   0    (data)    gate_ID#-pin_pathname...
  ...
```

The list associated with the input and output pin names indicate the pins to which they are connected. For the primitive-level, this also includes the gate index number of the connecting gate and only includes the pin pathname if one exists at
that point. There is a limitation on reporting gates at the design-level. If some circuitry inside the design cell is completely isolated from other circuitry, the command only reports the circuitry associated with the pin pathname.

You can change the output of the Report Gates command by using the Set Gate Report command.

You must flatten the netlist before issuing this command.

**Reporting on the First Input of a Gate**

Report Gates provides a shortcut to display data on the gate connected to the first input of the previously reported gate. This lets you quickly and easily trace backward through circuitry. To use Report Gates in this manner, first report on a specific gate and then enter:

```
SETUP> b
```

The following example shows how to use Report Gate and B commands to trace backward through the first input of the previously reported gate.

```
SETUP> rep gate 26
   // /u1/inst__565_ff_d_1__13 (26) BUF
   //   "I0"   I 269-
   //   "OUT"  O 268- 75-

SETUP> b
   // /u1/inst__565_ff_d_1__13 (269) LA
   //   "S"    I 14-
   //   "R"    I 145-
   //   SCLK   I 4-/clk
   //   D      I 265-/u1/_g32/X
   //   ACLK   I 2-/scan_mclk
   //   SDI    I 20-/u1/inst__565_ff_d_0__dff/Q2
   //   "OUT"  O 26- 27-

SETUP> b
   // /u1/inst__565_ff_d_1__13 (14) TIE0
   //   "OUT"  O 269- 268-
```
Reporting on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also use a shortcut to trace forward through the first fanout of the previously reported gate. To use Report Gates in this manner, first report on a specific gate and then enter:

```
SETUP> f
```

The following example shows how to use Report Gate and F commands to trace forward through the first fanout of the previously reported gate.

```
SETUP> rep ga 269
// /u1/inst__565_ff_d_1__13 (269)  LA
// "S"  I  14-
// "R"  I  145-
// SCLK  I  4-/clk
// D    I  265-/u1/_g32/X
// ACLK  I  2-/scan_mclk
// SDI   I  20-/u1/inst__565_ff_d_0__dff/Q2
// "OUT" O  26-  27-
```

```
SETUP> f
// /u1/inst__565_ff_d_1__13 (26)  BUF
// "I0"  I  269-
// "OUT" O  268-  75-
```

```
SETUP> f
// /u1/inst__565_ff_d_1__13 (268)  LA
// "S"  I  14-
// "R"  I  145-
// BCLK  I  1-/scan_sclk
// "D0"  I  26-
// "OUT" O  24-  25-
```

### Arguments

- **gate_id#**

  A repeatable integer that specifies the gate identification numbers of the objects for which you want to display gate information. The value of the `gate_id#` argument is the unique identification number that DFTAdvisor automatically assigns to every gate within the design during the model flattening process.
• **pin_pathname**
  A repeatable string that specifies the names of pins within the design for which you want to display gate information.

• **instance_name**
  A repeatable string that specifies the top-level boundary instance names within the design for which you want to display gate information. This is used for the design-level only.

• **-Type gate_type**
  A repeatable switch and name pair that specifies the gate types for which you want to display the gate information. The supported gate_types are listed in Table 2-3.

### Table 2-3. Report Gate Types

<table>
<thead>
<tr>
<th>gate_type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUF</td>
<td>buffer</td>
</tr>
<tr>
<td>INV</td>
<td>inverter</td>
</tr>
<tr>
<td>AND</td>
<td>and</td>
</tr>
<tr>
<td>NAND</td>
<td>inverted and</td>
</tr>
<tr>
<td>OR</td>
<td>or</td>
</tr>
<tr>
<td>NOR</td>
<td>inverted or</td>
</tr>
<tr>
<td>XOR</td>
<td>exclusive-or</td>
</tr>
<tr>
<td>XNOR</td>
<td>inverted exclusive-or</td>
</tr>
<tr>
<td>DFF</td>
<td>D flip-flop, same as _dff library primitive</td>
</tr>
<tr>
<td>LA</td>
<td>latch, same as _dlat library primitive</td>
</tr>
<tr>
<td>PI</td>
<td>primary input</td>
</tr>
<tr>
<td>PO</td>
<td>primary output</td>
</tr>
<tr>
<td>TIE0</td>
<td>tied low</td>
</tr>
<tr>
<td>TIE1</td>
<td>tied high</td>
</tr>
<tr>
<td>TIEX</td>
<td>tied unknown</td>
</tr>
</tbody>
</table>
Examples

The following example displays the simulated values of the gate and its inputs.

```
set system mode dft
set gate report error_pattern
set gate level primitive
report gates i_1006/o
/P2.13P (20) NAND
   A   I   10-/LD.1
   B   I   7-/M1.1
   Z   O   30-/P2.2P/S
```

The gate report for the design level may look like the following:

```
/P2.13P ND2
   A   I   /LD.1
   B   I   /M1.1
   Z   O   /P2.2P/S
```
Related Commands

Set Gate Level

Set Gate Report
Report LFSR Connections

Scope: All modes

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

REPort LFsr Connections

Description

Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.

The Report LFSR Connections command displays all of the connections between the LFSRs and the primary pins. These connections were specified with the Add LFSR Connections commands.

Examples

The following example displays the connections between the LFSRs and primary pins:

```
add lfsrs lfsr1 prpg 5 15 -serial -in
add lfsrs lfsr2 prpg 5 13 -serial -in
add lfsrs misr1 misr 5 11 -parallel -in
add lfsr taps lfsr1 1 3
add lfsr taps lfsr2 2 3
add lfsr connections scan_in.1 lfsr1 3
add lfsr connections scan_out.0 misr1 2
report lfsr connections
```

Related Commands

Add LFSR Connections
Delete LFSR Connections
Setup LFSRs
Report LFSRs

Scope: All modes

Prerequisites: You must have defined LFSRs with the Add LSFRs command. This command is intended for an LBISTArchitect design flow.

Usage

REPort LFsr

Description

Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).

The Report LFSRs command displays all of the LFSRs with their current values and tap positions, which you specified using the Add LFSRs and Add LFSR Taps commands.

You use this command primarily when implementing Built-In Self-Test (BIST) circuitry to display the calculated signature.

Examples

The following example displays the definitions of all the current LFSRs:

```
add lfsrs lfsr1 prpg 5 15 -serial -in
add lfsrs lfsr2 prpg 5 13 -serial -in
add lfsrs misr1 misr 5 11 -parallel -in
report lfsrs
```

Related Commands

Add LFSRs
Add LFSR Taps
Delete LFSRs
Setup LFSRs
Report Loops
Scope: Bist and Dft mode

Usage
REPort LOops

Description
Displays a list of all the current loops.

The Report Loops command displays all the loops in the circuit. For each loop, the report indicates whether the loop was broken by duplication. Loops that are not broken by duplication are shown as being broken by a constant value, which means the loop is either a coupling loop or has a single multiple fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

You can write the loops report information to a file by using the Write Loops command.

Examples
The following example displays a list of all the loops in the circuit:

```
set system mode dft
report loops
```

Related Commands
Write Loops
Report Mapping Definition

Scope: Setup and DFT modes

Usage

REPort MApping Definition -All | {object_name [ -Instance | -Module]}
[ -Nonscan_model nonscan_model_name] [-Scan_model scan_model_name]
[ -Output scan_output_pin_name]} [-Filename filename [-Replace]]

Description

Reports the nonscan to scan model mapping defined in the design.

The Report Mapping Definition command reports the mapping of nonscan models to scan models. You can report the scan and output mapping for an individual instance, all instances under a hierarchical instance, all instances in all occurrences of a module in the design, all occurrences of the model in the entire design, or the entire design.

The report is always displayed in the transcript. You can optionally write it to a file using the -Filename switch.

Arguments

- **-All**
  A required switch that specifies to report all scan and output mapping in the entire design.

- **object_name**
  A required string that specifies the name of the nonscan model you want to report on. You can also specify an instance, hierarchical instance, module, or scan model.

  - If this argument is the name of an instance or hierarchical instance, the -Instance switch is required and optionally the model can be specified with the -Nonscan_model switch or -Scan_model switch.

  - If this argument is the name of a module, then the -Module switch is required and optionally the model can be specified with the -Nonscan_model or -Scan_model switch.
If this argument is a scan model, then the -Output switch is required. Since you specified a scan model, you can only report the scan output pin mapping.

-Instance | -Module

An optional switch that specifies the type of the object_name argument. If neither switch is specified, the object_name is a model (the default).

- If you specify -Instance and the instance is primitive, then only the named instance is reported.

- If you specify -Instance and the instance is hierarchical, then all instances under that instance are reported. Optionally, you can constrain the report to matching the -Nonscan_model or (for output mapping) matching the -Scan_model.

- If you specify -Module, then for all occurrences of that module, all instances within that module are reported. Optionally, you can constrain the report to matching the -Nonscan_model or (for output mapping) matching the -Scan_model.

-Nonscan_model nonscan_model_name

A switch and string pair that specifies the name of the nonscan model that you want to report on. This argument is required only if you specify -Instance or -Module switch and want to constrain the report to objects matching the nonscan model, otherwise, you can specify the nonscan model in the object_name argument.

-Scan_model scan_model_name

A switch and string pair that specifies the name of the scan model to report on. This argument is required when you want to further constrain the report, except when you are only reporting the mapping of the scan output pin and specify the scan model in the object_name argument.

-Output [scan_output_pin_name]

An optional switch and string pair that specifies the name of the scan output pin. You can use this to constrain the report. Specifying just the -Output switch reports all mapped scan output pins for the specified scan model, while
specifying the switch with a pin name reports the mapping for only scan models that use that pin for the scan output.

- **Filename filename [-Replace]**
  
  An optional switch and string that specifies that DFTAdvisor writes the scan mapping report to a file. The -Replace switch specifies that the file should be overwritten if it already exists.

### Examples

The following example reports the scan and output mapping for all occurrences the fd1 nonscan model in the design:

```
report mapping definition fd1
```

The following example reports the mapping for each occurrence of the fd1 nonscan model mapped to the fd1s scan model with the scan output pin mapped to “qn”:

```
report mapping definition fd1 -scan_model fd1s -output qn
```

The following example reports the mapping for each occurrence of the fd1s scan model mapped to the fd1s scan model with the scan output pin mapped to “qn” for all matching instances in the “counter” module and for all occurrences of that module in the design:

```
report mapping definition counter -module -scan_model fd1s -output qn
```

### Related Commands

- Add Mapping Definition
- Delete Mapping Definition
Report Mos Direction

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

REP ort MOs Direction [-Unidirection | -Bidirection | -All]

Description

Reports the direction MOS instances in the Spice design and Spice SUBCKT library.

The Report Mos Direction command reports the direction of all or specified MOS instances in the Spice design and library. By default, only the bi-directional MOS instances are reported.

Arguments

- -Unidirection
  An optional switch that specifies to list all MOS instances that have a defined direction.

- -Bidirection
  An optional switch that specifies to list all MOS instances that have a no defined direction (bi-directional). This is the default.

- -All
  An optional switch that specifies to list all directional and bi-directional MOS instances.

Examples

The following example reports all bi-direction MOS instances:

  report mos direction

Related Commands

Add Mos Direction    Delete Mos Direction
Report Mtpi Controller

Scope: All modes
Prerequisites: This command intended for an LBISTArchitect design flow.

Usage
REPort MTpi Controller controller_name | -All

Description
Reports the state data related to the MTPI controller(s).
The REPort Mtpi Controller command lists the state data for the specified or all MTPI controllers in the design.

Arguments
- controller_name
  A string that specifies the name of the MTPI controller to report on.
- -All
  A switch that specifies to report on all defined MTPI controllers.

Examples
The following example lists the state data for the “controller1” MTPI controller:

report mtpi controller controller1

MTPI Controller controller1 Connections =
  corecomp/core_i/phase_1 corecomp/core_i/phase_2
  corecomp/core_i/phase_3
Output_states:
cycle = 0 output = 001
cycle = 2 output = 010
cycle = 10 output = 100
cycle = 18 output = 000
Related Commands

Add Mtpi Controller  Delete Mtpi Output
Add Mtpi Output  Set Bist Initialization
Delete Mtpi Controller
Report Net Properties

Scope: Setup mode

Prerequisites: This command can only operate on a Spice design.

Usage

REPORT NET Properties {-VDD | -GND | -All}

Description

Reports the VDD or GND net properties in the Spice design and library.

The Report Net Properties command reports all Spice VDD or GND net properties in the Spice design and Spice library. You can also specify to report all of the VDD and GND nets at one time using the -All option.

Arguments

• -VDD | -GND | -All
  A required switch that specifies whether to report VDD or GND net properties. You can also specify to report on both using the -All switch.

Examples

The following example reports all GND nets:

  report net properties -gnd

Related Commands

Add Net Property  Delete Net Property
Report Nofaults

Scope: All modes

Usage

REPort NOfaults {pathname... | -All} [-Instance] [-Stuck_at {01 | 0 | 1}]

Description

Displays the no-fault settings for the specified pin or instance pathnames.

The Report Nofaults command displays for pin pathnames or pin names of instances the nofault settings which you previously specified with the Add Nofaults command.

Arguments

- **pathname**
  A repeatable string that specifies the pin pathnames or the instance pathnames for which you want to display the nofault settings. If you specify an instance pathname, you must also specify the -Instance switch.

- **-All**
  A switch that specifies to display the nofault settings on either all pin pathnames or, if you also specify the -Instance switch, all pin names of instances.

- **-Instance**
  An optional switch that specifies that the pathname or -All argument indicates instance pathnames.

- **-Stuck_at 01 | 0 | 1**
  An optional switch and literal pair that specifies the stuck-at nofault settings which you want to display. The valid stuck-at literals are as follows:

  - **01** — A literal that specifies to display both the “stuck-at-0” and “stuck-at-1” nofault settings. This is the default.
  - **0** — A literal that specifies to only display the “stuck-at-0” nofault settings.
  - **1** — A literal that specifies to only display the “stuck-at-1” nofault settings.
Examples

The following example displays all pin names of the instances that have the nofault settings:

```
add nofaults i_1006 i_1007 i_1008 -instance
report nofaults
```

Related Commands

Add Nofaults
Delete Nofaults
Report Nonscan Instances

Scope: All modes

Usage

REPort NONscan Instances [-Class {Full | System | User}]

Description

Displays the currently defined sequential non-scan instances.

The Report Nonscan Instances command displays the sequential non-scan instances which either you added by using the Add Nonscan Instances command, or that have the Dont_touch property in a Genie netlist.

Arguments

• -Class Full | System | User

An optional switch and literal pair that specifies the source (or class) of the sequential non-scan instances which you want to display. The valid literals are as follows:

  Full — A literal that specifies to display all the non-scan sequential instances in the user and system class. This is the default.

  User — A literal that specifies to only display the non-scan sequential instances that are the result of the Add Nonscan Instances command.

  System — A literal that specifies to only display the non-scan sequential instances that are a result of the Genie netlist containing the Dont_touch property.

Examples

The following example displays all sequential non-scan instances that you added to the non-scan instance list:

  set system mode dft
  add nonscan instances i_1006 i_1007 i_1008
  report nonscan instances -class user

  USER:   i_1006
  USER:   i_1007
  USER:   i_1008
Related Commands

Add Nonscan Instances  Delete Nonscan Instances
Report Nonscan Models

Scope: All modes

Usage

REPORT NONscan Models [-Class {Full | System | User}]

Description

Displays the sequential non-scan model list.

The Report Nonscan Models command displays sequential models which either you added by using the Add Nonscan Models command, or that have the Dont_touch property in a Genie netlist.

Arguments

- -Class Full | System | User

An optional switch and literal pair that specifies the source (or class) of the sequential non-scan models which you want to display. The valid literals are as follows:

- **Full** — A literal that specifies to display all the non-scan sequential models in the user and system class. This is the default.
- **User** — A literal that specifies to only display the non-scan sequential models that are the result of the Add Nonscan Models command.
- **System** — A literal that specifies to only display the non-scan sequential models that are a result of the Genie netlist containing the Dont_touch property.

Examples

The following example displays all sequential non-scan models from the non-scan model list:

```
set system mode dft
add nonscan models d_flip_flop1 d_flip_flop2
report nonscan models
```
Related Commands

Add Nonscan Models    Delete Nonscan Models
Report Notest Points

Scope: DFT mode

Usage

REPort Notest Points -Paths

Description

Displays all the circuit points for which you do not want DFTAdvisor to insert controllability and observability.

The Report Notest Points command displays the circuit points added using the Add Notest Points command and which therefore, DFTAdvisor cannot use for testability insertion. You can also list the critical path definitions that added notest points by using the -Paths switch.

You use this command primarily when implementing Built-In Self-Test (BIST) circuitry.

Arguments

- -Paths

An optional switch that displays the definitions for all currently loaded critical paths that have marked notest points. If this switch is not specified, only the resulting notest points are reported.

Examples

The following example displays the list of all circuit points that DFTAdvisor cannot use for testability insertion:

```
set system mode dft
add notest points tr_io
add notest points ts_i
report notest points
```

Related Commands

Add Notest Points  Delete Notest Points
**Report Output Masks**

Scope: All modes

**Usage**

REPort OUtput Masks

**Description**

Displays a list of the currently masked primary output pins.

The Report Output Masks command displays the primary output pins that you previously masked by using the Add Output Masks command. When you mask a primary output pin, you inform DFTAdvisor to mark that pin as an invalid observation point during the scan cell identification process. DFTAdvisor uses all unmasked primary output pins as possible observation points to which the effects of all faults propagate for detection.

**Examples**

The following example masks two primary outputs and then displays the results:

```
add output masks q1 -hold1
add output masks qb1 -hold 0
report output masks
  q1  hold1
  qb1 hold0
```

**Related Commands**

Add Output Masks  Delete Output Masks
Analyze Output Observe  Setup Output Masks
**Report Pin Constraints**

Scope: All modes

**Usage**

REPort PIN Constraints [-All | primary_input_pin...]

**Description**

Displays the pin constraints of the primary inputs.

The Report Pin Constraints command displays the pin constraints that you previously added to the primary inputs with the Add Pin Constraints command. You can change the constant value constraints of the primary inputs by using the Add Pin Constraints or Setup Pin Constraints commands.

![Note]

This reported information from this command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

**Arguments**

- **-All**
  
  An optional switch that specifies to display the current constraints for all primary input pins. This is the default.

- **primary_input_pin**
  
  An optional repeatable string that specifies a list of primary input pins whose constraints you want to display.

**Examples**

The following example displays the cycle behavior constraints of all primary inputs.

```
add pin constraints ph1 c0
add pin constraints ph2 c1
report pin constraints -all
```
Related Commands

Add Pin Constraints      Setup Pin Constraints
Delete Pin Constraints   Setup Scan Identification
Report Pin Equivalences

Scope: All modes

Usage
REPort PIN Equivalences

Description
Displays the pin equivalences of the primary inputs.
The Report Pin Equivalences command displays a list of primary inputs which you previously restricted to be at equivalent or complementary values by using the Add Pin Equivalences command.

Note
The reported information from this command has effects on other commands that relate to fault simulation; this includes simulation-based and multiphase test points selection, along with BIST pattern simulation (in BIST mode).

Examples
The following example displays all pin equivalences that have been added to the primary inputs:

```
add pin equivalences indata2 indata4
add pin equivalences indata3 -invert indata5
report pin equivalences
```

Related Commands
Add Pin Equivalences Delete Pin Equivalences
Report Primary Inputs

Scope: All modes

Usage

REPort PRimary Inputs [-All | primary_input_pin...]

Description

Displays the specified primary inputs.

The Report Primary Inputs command displays a list of either all the primary inputs of a circuit or a specific list of primary inputs that you specify.

Arguments

• -All
  An optional switch that specifies to display all the primary inputs. This is the default.

• primary_input_pin
  An optional repeatable string that specifies a list of primary input pins that you want to display.

Examples

The following example displays all of the primary inputs:

report primary inputs -all
SYSTEM:    /clk
SYSTEM:    /datain

Note

The label “system” means that these are primary inputs that DFTAdvisor automatically recognizes because they were in the netlist. Because there is no Add Primary Input command in DFTAdvisor as there is in FastScan and FlexTest, all primary inputs are of the system-defined class.

Related Commands

Write Primary Inputs
Report Primary Outputs

Scope: All modes

Usage

REPort PRimary Outputs [-All | primary_output_pin...]

Description

Displays the specified primary outputs.

The Report Primary Outputs command displays a list of either all the primary outputs of a circuit or a specific list of primary outputs that you specify.

Arguments

- **-All**
  
  An optional switch that specifies to display all the primary outputs. This is the default.

- **primary_output_pin**
  
  An optional repeatable string that specifies a list of primary output pins that you want to display.

Examples

The following example displays all of the primary outputs:

```plaintext
report primary outputs -all
SYSTEM:    /dataout
SYSTEM:    /dataout1
```

\[\text{Note}\]

The label “system” means that these are primary outputs that DFTAdvisor automatically recognizes because they were in the netlist. Because there is no Add Primary Output command in DFTAdvisor as there is in FastScan and FlexTest, all primary outputs are of the system-defined class.

Related Commands

Write Primary Outputs
Report Read Controls

Scope: All modes

Usage

REPort REad Controls

Description

Displays all of the currently defined read control lines.

The Report Read Controls command displays all the read control lines that you previously specified by using the Add Read Controls command. The display also includes the corresponding off-state with each read control line.

Examples

The following example displays a list of the current read control lines:

```
add read controls 0 r1 r3
add read controls 1 r2 r4
report read controls
```

Related Commands

Add Read Controls  Delete Read Controls
**Report Scan Cells**

Scope: Dft mode

**Usage**

REPort SCan CElls [-All | chain_name...] [-Filename filename [-Replace]]

**Description**

Displays a report or writes a file on the scan cells that reside in the specified scan chains.

The Report Scan Cells command provides a report on the scan cells within specific scan chains. The following information is provided in the report for each scan cell:

- Chain cell index number (where 0 is the scan cell closest to the scan-out pin)
- Scan chain in which the scan cell resides (chain1, chain2,... are the default chain names if reporting scan cells on inserted scan chains)
- Scan group in which the scan chain resides (dummy is the default group name if reporting scan cells on inserted scan chains)
- Instance name of the scan cell

If you issue the command without specifying any arguments, then DFTAdvisor displays a report on the scan cells for all scan cells in existing scan chains and also scan cells from the inserted scan chains.

If you issue the command with the -Filename switch, then DFTAdvisor writes the scan cells to a file in the format that can be read by the Insert Test Logic command. The format of the written file is different than the format of the viewed report. You can optionally edit the scan cell order in the file before reading the file with the Insert Test Logic command.
Report Scan Cells

Command Dictionary

Arguments

- `-All | chain_name...`
  
  An optional switch or repeatable string. The -All switch specifies to display the scan cells for all scan chains. This is the default. The repeatable string specifies the scan chains whose scan cells you want to display.

- `-Filename filename [-Replace]`
  
  An optional switch and string that specifies that DFTAdvisor writes the list of scan cells to a file. The format of the written file is different than the format of the viewed report. The -Replace switch specifies that the file should be overwritten if it already exists.

Examples

The following example displays a list of all scan cells, in the DFT system mode:

```
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 outdata4
set system mode dft
report scan cells
  0 chain1 group1 /MQ_I400
  1 chain1 group1 /FH_I400
  2 chain1 group1 /FQ_I10
  3 chain1 group1 /RP_I10
  4 chain1 group1 /IS_I10
  5 chain1 group1 /CZ_I400
```

The first column displays the chain cell index number, where 0 is the scan cell closest to the scan-out pin.

The second column displays the chain name where the scan cell resides.

The third column displays the group name where the scan cell resides.

The fourth column displays the gate name of the memory element.

Related Commands

Add Scan Chains  Add Scan Groups
Report Scan Chains

Scope: All modes

Usage

REPort SCan CHains

Description

Displays a report on all the current scan chains.

The Report Scan Chains command provides the following information in a report for each scan chain:

- Name of the scan chain
- Name of the scan chain group
- Scan chain input and output pins
- Length of the scan chain

Examples

The following example displays a report of all the scan chains:

```
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 outdata4
add scan chains chain2 group1 indata3 outdata5
report scan chains
```

Related Commands

Add Scan Chains  Delete Scan Chains
Report Scan Groups

Scope: All modes

Usage
REPort SCan Groups

Description
Displays a report on all the current scan chain groups.
The Report Scan Groups command provides the following information in a report for each scan chain group:

- Name of the scan chain group
- Number of scan chains within the scan chain group
- Number of shifts
- Name of the test procedure file, which contains the information for controlling the scan chains in the reported scan chain group

Examples
The following example displays a report of all the scan groups:

```
add scan groups group1 scanfile
add scan groups group2 scanfile1
report scan groups
```

Related Commands
Add Scan Groups                      Delete Scan Groups
Report Scan Identification

Scope: All modes

Usage

REPort SCan IDentification [ -Full | -Identified | -Defined ]

Description

Displays a list of the scan instances which DFTAdvisor has identified or you have defined as scan cells.

The Report Scan Identification command displays scan cell instances that either DFTAdvisor identified during the identification process or that you defined by using the Add Scan Instances or Add Scan Models commands. The Report Scan Identification command lists the scan instances in descending order, with the first instance being the most critical scan instance.

If you are identifying scan sequential, the Report Scan Identification command displays the sequential loops that DFTAdvisor cut after you have performed the identification process with the Run command.

If you are identifying partition scan, the Report Scan Identification command displays the partition cells that DFTAdvisor flagged during the identification process that you perform with the Run command.

If you issue the command without specifying any arguments, then the command displays both identified and defined scan instances.

Arguments

- -Full
  An optional switch that specifies to display both identified and defined scan instances. This is the default.

- -Identified
  An optional switch that specifies to display only scan instances that DFTAdvisor identified during the identification process.
• **-Defined**
  
  An optional switch that specifies to display only scan instances that you defined by using the Add Scan Instances or Add Scan Models commands.

**Examples**

The following example displays all scan instances after performing a full scan identification run:

```bash
set system mode dft
setup scan identification full_scan
run
report scan identification -identified
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>defined</td>
<td>/I_1_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_1_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_6_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_8_16</td>
</tr>
</tbody>
</table>

**Related Commands**

- Setup Scan Identification
- Write Scan Identification
Report Scan Instances

Scope: All modes

Usage

REPort SCAn Instances [-Class {Full | User | System}]

Description

Displays the currently defined sequential scan instances.

The Report Scan Instances command displays the sequential scan instances which either you added by using the Add Scan Instances command, or DFTAdvisor identified as being within a scan chain.

Arguments

- -Class Full | User | System

An optional switch and literal pair that specifies the source (or class) of the sequential scan instances which you want to display. The valid literals are as follows:

Full — A literal that specifies to display all the scan sequential instances in the user and system class. This is the default.

User — A literal that specifies to only display the scan sequential instances entered by the user.

System — A literal that specifies to only display the scan sequential instances described in the netlist.

Examples

The following example displays all sequential scan instances from the scan instance list.

add scan instances i_1006 i_1007 i_1008
report scan instances

Related Commands

Add Scan Instances     Delete Scan Instances
Report Scan Models

Scope: All modes

Usage

REPort SCan Models

Description

Displays the sequential scan models currently in the scan model list.

The Report Scan Models command displays sequential models which you previously added to the scan model list by using the Add Scan Models command.

Examples

The following example displays all the sequential scan models from the scan model list:

```
set system mode dft
add scan models d_flip_flop1 d_flip_flop2
report scan models
```

Related Commands

Add Scan Models          Delete Scan Models
Report Scan Pins

Scope: All modes

Usage

REPort SCan Pins

Description

Displays all previously assigned scan input, output, and clock names.

The Report Scan Pins command displays the user-specific names that you have added to the scan input and scan output pins of the scan chains.

Examples

The following example displays all scan input and scan output names for the scan chains and then inserts the scan chains.

```
add clocks 0 clk
set system mode dft
run
add scan pins chain1 si so
add scan pins chain2 si1 so1
report scan pins
insert test logic -max_length 5
```

Related Commands

Add Scan Pins Delete Scan Pins
Report Seq_transparent Constraints

Scope: All modes

Usage

REPort SEq_transparent Constraints

Description

Displays the seq_transparent constraints.

The Report Seq_transparent Constraints command displays the constraints that you added to DFT library model clock enable pins with the Add Seq_transparent Constraints command. You can change the constant value constraints of the pins by using the Add Seq_transparent Constraints command.

Examples

The following example displays all the seq_transparent constraints.

```
report seq_transparent constraints
```

Related Commands

Add Seq_transparent Constraints     Setup Scan Identification
Delete Seq_transparent Constraints
Report Statistics

Scope: All modes

Usage

REPort STAtistics

Description

Displays a detailed report of the design’s statistics.

The Report Statistics command displays a detailed statistics report to the screen. The report includes the following information when in Setup and Dft modes:

- Total number of sequential instances
- Number of defined non-scan instances
- Number of non-scan instances identified by the DRC
- Number of defined scan instances
- Number of scan instances identified by the DRC
- Number of identified scan instances
- Number of scannable instances with test logic
- Number of preexisting scan chains

The report includes the following fault simulation information when in Bist mode:

- The current number of collapsed and total faults in each class. The report does not display fault classes with no members.
- The percentage of test coverage, fault coverage, and ATPG effectiveness for both collapsed and total faults
The total numbers for the following:

- Total patterns simulated in the preceding fault simulation process. This subgroup may additionally contain total numbers for the following internal patterns sets:
  - basic scan patterns
  - Clock_po patterns
  - Ram_sequential patterns
  - Clock_sequential patterns

- Total patterns currently in the test pattern set

- Total CPU time.

If a pattern type has no patterns, the report does not display the count for that type. If all patterns are basic patterns, it will not display any count. An it counts clock_sequential patterns that are also clock_po only as clock_sequential patterns.

**Examples**

The following example displays the statistics report after performing the scan identification process in Dft mode:

```
add clocks 0 clock
set system mode dft
run
report statistics
Total number of sequential instances   =40
Number of defined nonscan instances   =5 (12.50%)
Number of nonscan instances identified by drc =5 (12.50%)
Number of defined scan instances      =5 (12.50%)
Number of scan instances identified by drc =5 (12.50%)
Number of identified scan instances   =5 (12.50%)
Number of scannable instances         =10
Number of scannable instances with test logic =5
```
Report Sub Chains

Scope: All modes

Usage

REPort SUb Chains

Description

Generates and displays a report on the scan sub-chains.

The Report Sub Chains command transcripts the scan sub-chain definition(s), which includes the scan type, module or instance pathname, name of the sub-chain, defined length of the sub-chain, scan input of the sub-chain, and the scan output of the sub-chain.

For mux-scan the report includes the scan enable of the sub-chain.

For clocked-scan the report includes the scan clock.

For LSSD the report includes the scan master clock and the scan slave clock.

Examples

The following example generates a report for a scan sub-chain:

```
add sub chains sub1 subc1 /subscan_in1 /subscan_out1 10 mux_scan
/subscan_en -module
```

```
report sub chains
mux_scan: SUB1 subc1 10 subscan_in1 subscan_out1 subscan_en
```

Related Commands

Add Sub Chains  Delete Sub Chains
Report Test Logic

Scope: All modes

Prerequisites: You must first use the Insert Test Logic command to add the test logic or test points.

Usage

REPORT TEST LOGIC [-Instance | -Module | -Summary | -Location]

Description

Displays the test logic that DFTAdvisor added during the scan insertion process.

The Report Test Logic command displays information about the test logic that DFTAdvisor added during the scan insertion process as a result of the Set Test Logic and Add Test Points commands.

Arguments

- **-Instance**
  
  An optional switch that displays the list of instance pathnames and the corresponding DFT library model that DFTAdvisor inserted as test logic and test points. This option also includes a summary of the number of each instance-based DFT library model that DFTAdvisor inserted. This is the default.

- **-Module**
  
  An optional switch that displays the list of module names, the list of instance pathnames, and the corresponding DFT library models that DFTAdvisor inserted as test logic and test points. This option also includes a summary of the number of each module-based DFT library model that DFTAdvisor inserted.

- **-Summary**
  
  An optional switch that displays a summary that contains the number of each module and instance-based DFT library model that DFTAdvisor inserted.
• -Location

  An optional switch that displays the pin pathname where DFTAdvisor inserted test logic and identifies whether it was a result of the test logic settings or test points that you specified.

Examples

The following example uses both test logic and test points. The report displays the locations where DFTAdvisor inserted the test logic as a result of both the Add Test Point command and the Set Test Logic command:

  add cell models and2a -type and
  add cell models inv1a -type inv
  add cell models mux1a -type mux s a b
  add test point /I_6_16/cp control and2a control_input
  set test logic -set on -reset on
  set system mode dft
  run
  insert test logic
  report test logic -location
  /I_6_16/reset (test points)
  /I_7_16/set   (scan cell)

Related Commands

  Add Test Points                          Report Test Points
  Delete Test Points
Report Test Points

Scope: All modes

Usage

REPort TEst Points [-Full | -Control | -Observe | -Lockup]

Description

Displays the test point specifications you created with Add Test Points command and any test points that you enabled DFTAdvisor to automatically identify.

The Report Test Point command displays the test point specifications for both user-defined and system-defined test points. User-defined test points are those that you created by using the Add Test Point command. However, DFTAdvisor does not actually generate the test point circuitry until you issue the Insert Test Logic command.

System-defined test points are those that DFTAdvisor automatically identifies and inserts based on the information you provide with the following commands:

- Setup Scan Identification
- Setup Test_point Identification
- Run
- Setup Test_point Insertion
- Insert Test Logic

The report marks the system-defined test points with “[Selected]” prior to the test point information.

Arguments

- -Full

An optional switch that specifies to display all of the information available on both control and observe test points for both the user-defined and the system-defined test points. This is the default.
- Control
An optional switch that specifies to display all the test point definitions (both user- and system-defined) that are for the purpose of enabling better test coverage in design areas where, previously, DFTAdvisor could not force certain state values.

- Observe
An optional switch that specifies to display all the test point definitions (both user- and system-defined) that are for the purpose of enabling better test coverage by allowing the tester access to certain fault effects.

- Lockup
An optional switch that specifies to display all test points (both user- and system-defined) with added lockup latches.

Examples
The following example creates one user-defined control point and one user-defined observe point and then reports their definitions:

```
add test point /I_7_16/q observe_output
add cell models and2a -type and
add cell models sdff1a -type sdff clk data
add test point /I_6_16/reset control and2a control_input -intscan sdff1a
report test points
Control: /I_6_16/reset and2a control_input sdff1a (internal scan)
Observe: /I_7_16/q observe_output
```

Related Commands
- Add Test Points
- Delete Test Points
- Report Test Logic
- Setup Scan Identification
- Setup Test_point Identification
Report Testability Analysis

Scope: Dft mode

Prerequisites: You must first issue the Analyze Testability command to calculate the values on which to report.

Usage

REPort TEstability Analysis [pathname] [-Controllability | -OBservability] [{-Number integer} | {-Percent integer} | {-OVer integer}]

Description

Displays the results of the Analyze Testability command.

The Report Testability Analysis command displays a columnar list of either the controllability or observability values for each pin in the flattened design. The -Controllability and -Observability switches determine the column definitions.

When this command is run from within the graphical user interface, the output (which can be very large) is sent into a separate window. The output is not sent to the session transcript window for performance reasons.

The Analyze Testability command calculates the controllability and observability values for each gate in the flattened design. If the design’s fault coverage in FastScan or FlexTest is lower than you desire, you can re-invoke DFTAdvisor to perform the testability analysis, which allows you access to the controllability and observability values. You can then generate test points (either manually or automatically) based on the results of the testability analysis to help increase the design’s fault coverage.

If you are going to manually investigate the results of the testability analysis and insert user-defined test points, you need to use the Add Test Points command. If you are going to have DFTAdvisor automatically identify (system-defined) test points, you need to use a combination of the Setup Scan Identification, Setup Test_point Identification, and Run commands.

For more information on inserting test points, refer to the Scan and ATPG Process Guide.
Arguments

- **pathname**
  
  An optional string that specifies the instance name whose pins for which you want DFTAdvisor to display the controllability or observability values. The default is all pins for all instances.

- **-Controllability**
  
  An optional switch that specifies for DFTAdvisor to only display the pin controllability values. This is the default. The controllability report displays the following information in columnar format:
  
  o The controllability value for the low logic state
  o The controllability value for the high logic state
  o The primitive gate type
  o The gate identification number
  o The pathname to the gate
  
  If DFTAdvisor cannot control the inputs of a gate, the report displays NC (non-controllable) for the corresponding logic state.

- **-OBservability**
  
  An optional switch that specifies for DFTAdvisor to only display the pin observability values. The observability report displays the following information in columnar format:
  
  o The observability value
  o The primitive gate type
  o The gate identification number
  o The pathname to the gate
  
  If DFTAdvisor cannot observe the outputs of a gate at any observation point, the report displays NO (non-observable).

- **-Number integer**
  
  An optional switch and integer pair that specifies the maximum number of pins whose values you want to display. If you specify the -Number switch, you must provide the associated integer.
• -Percent integer
  
  An optional switch and integer pair that specifies the percentage of the total number of available design pins whose values you want to display. You determine the total number of available design pins by whether you specify or do not specify the instance *pathname* argument. If you specify the -Percent switch, you must provide the associated *integer*.

• -Over integer
  
  An optional switch and integer pair that specifies the minimum controllability or observability values whose pins you want to display. If you specify the -Over switch, you must provide the associated *integer*.

**Examples**

The following example displays the controllability values of five percent of all the pins in the design.

```plaintext
set system mode dft
setup scan identification none
analyze testability -scoap_only
setup test_point identification -control 1
run
// Performing test_point identification ...
// Number of control points to be identified = 1
// Number of observe points to be identified = 0
// 1: CV1=16458417  gate_index=3805  INV /CNTR/U783/ZN
report testability analysis -controllability -percent 5
NC          0          BUF          25        /I_6_16
0           NC         INV          27        /I_7_14
100         1          BUF          39        /I_8_21
```

The report displays the controllability value for the low logic state (where NC means non-controllable), the controllability value for the high logic state, the primitive gate type, the gate identification number, and the pathname to the gate.

**Related Commands**

- Analyze Testability
**Report Tied Signals**

Scope: All modes

**Usage**

`REPORT TIED SIGNALS [-CLASS {Full | User | System}]`

**Description**

Displays a list of the tied floating signals and pins.

The `REPORT TIED SIGNALS` command displays either the user class, system class, or full classes of tied floating signals and pins. If you do not specify a class, the command displays all the tied floating signals and pins.

**Arguments**

- `-CLASS {Full | User | System}`

  An optional switch and literal pair that specifies the source (or class) of the tied floating signals or pins which you want to display. The valid literals are as follows:

  - **Full** — A literal that specifies to display all the tied floating signals or pins in the user and system class. This is the default.
  - **User** — A literal that specifies to only display the tied floating signals or pins that you created by using the `ADD TIED SIGNALS` command.
  - **System** — A literal that specifies to only display the tied floating signals or pins described in the netlist.

**Examples**

The following example will display the tied signals from the user class.

```
ADD TIED SIGNALS 1 VCC VDD
REPORT TIED SIGNALS -CLASS USER
```

**Related Commands**

- Add Tied Signals
- Delete Tied Signals
- Setup Tied Signals
**Report Write Controls**

Scope: All modes

**Usage**

REPort WRite Controls

**Description**

Displays the currently defined write control lines and their off-states.

The Report Write Controls command displays the write control lines, with corresponding off-states, that you previously added by using the Add Write Controls command.

**Examples**

The following example adds four write control lines and then displays a list of the control line definitions:

```
add write controls 0 w1 w3
add write controls 1 w2 w4
report write controls
```

**Related Commands**

Add Write Controls  
Delete Write Controls
**Reset State**

Scope: All modes

**Usage**

RESet STate

**Description**

Removes all instances from both the scan identification and test point identification lists that DFTAdvisor identified during a run.

The Reset State command removes scan instances or test points identified with the Run command. However, if you have stitched the scan chain or inserted test points, this command has no effect.

**Examples**

The following example performs a full scan identification process, then removes the identified scan instances and performs a 75 percent ATPG scan identification process:

```
set system mode dft
setup scan identification full_scan
run
report scan identification
.
.
.
reset state
setup scan identification sequential atpg -percent 75
run
report scan identification
.
.
.
```
Ripup Scan Chains

Scope: Dft mode

Usage

RIPup SCAn Chains { -All | chain_name... } [-Output]

Description

Removes the specified scan chains from the design.

The Ripup Scan Chains command removes scan chains that DFTAdvisor placed during the scan insertion process. You can remove (rip up) either all the scan chains or individual scan chains. You can also rip up the scan chain output pin.

If you only wish to remove a scan chain definition that you previously created with the Add Scan Chains command, use the Delete Scan Chains command.

Note

- If the design contains test logic in addition to scan circuitry, this command only removes the scan circuitry, not the test logic.

Arguments

- **-All**
  A switch that specifies to remove all scan chains.

- **chain_name**
  A repeatable string that specifies the names of the scan chains that you want to remove.

- **-Output**
  An optional switch that specifies that the existing scan chain output pins are to be ripped up together with the scan chains.
Examples

The following example performs the scan insertion process with unlimited scan chain lengths, then remove those scan chains and re-runs the scan insertion process with a maximum scan chain length of 100:

```
add clocks 0 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
setup scan insertion -seb MY_SEL
insert test logic -nolimit
report scan chains
ripup scan chains -all
set system mode setup
set system mode dft
reset state
setup scan identification sequential atpg -percent 50
run
insert test logic -max_length 100
report scan chains
```
Run

Scope: Dft and Bist modes

Usage

RUN

Description

Runs the scan or test point identification process in Dft mode and runs the fault simulation and signature calculation process in Bist mode.

The Run command performs the scan or test_point identification process in Dft mode depending on the identification type you set with the Setup Scan Identification command. The Run command performs the scan identification process as indicated by the Setup Scan Identification command (if the identification type is set to -Sequential) and the test_point identification process as indicated by the Setup Test_point Identification command.

During the identification run, DFTAdvisor displays progress messages. The first number indicates the number of instances currently identified for scan (added to the scan candidate list). During the controllability phase, the second number indicates the estimated percentage of toggle coverage. During the observability phase, this number indicates the estimated observability coverage of stuck-at faults. For example, if you set the identification type to sequential, the following may be displayed for the controllability phase:

```
// Sequential instances identified = 238 (Controllability = 97.31%)
```

The Run command performs the fault simulation and signature calculation process in Bist mode. After the run, you can use the Report Statistics command to display the fault simulation results and use the Report Lfsrs command to display the calculated signature results.

The Run command does not reset the contents of LFSRs before beginning the run. You must use the Reset State command to reset the LFSRs.
Examples

The following example runs a scan identification process:

```plaintext
set system mode dft
setup scan identification sequential atpg
run
report scan identification
```

Related Commands

- Setup Scan Identification
- Setup Test_point Identification
Save Patterns

Scope: Bist mode

Prerequisites: You may use this command if the Set Pattern Save command is set to on.

Usage

SAVE PATterns filename [-Replace] -BIst_test [-TSSiwgl | -TSSIBinwgl | -Verilog | -VHdl] [-Mode Lsi] [timing_filename]

Description

Saves the current BIST test pattern set to a file in the format that you specify.

The Save Patterns command saves the current set of BIST patterns into a file. If the Set Pattern Save command is set to ON, DFTAdvisor stores BIST patterns during simulation. BIST patterns include the events that perform circuit initialization, patterns that “burst” the clock to pulse the BIST logic, and shift out the final MISR signature.

Once the circuit is set up to run BIST, the only thing needed to have the circuit test itself is to supply a number of clock pulses to the BIST logic. No other patterns need to be supplied to the circuit. So, instead of creating a large pattern set to supply a large number clock pulses, most simulation languages and test description languages (such as TSSI WGL) let you do use some sort of repeat or “burst” command to describe a large number of clock pulses in a single statement. This creates a much smaller pattern set.

The term “burst” is used because on most test equipment, a series of clock pulses can be run at a much higher frequency than general test patterns can. Therefore, it is described as providing a rapid “burst” of clock pulses.

Arguments

- **filename**
  A required string that specifies the name of the file to which you want to write the test pattern set.
- **Replace**
  An optional switch that specifies replacement of the contents of `filename`, if a file by that name already exists.

- **-B1st_test**
  A required switch that specifies to save the BIST patterns that were stored during BIST pattern simulation.

- **-TSSIWgl**
  A switch that specifies to write the patterns in the TSSI WGL format. The TSSI WGL format contains the waveform pattern information and any timing information from the timing file. You can use the TSSI WGL format patterns to generate test patterns in a variety of tester and simulator formats.

- **-TSSIBinwgl**
  A switch that specifies to write the patterns in the TSSI Binary WGL format. The TSSI Binary WGL format contains the waveform pattern information and any timing information from the timing file.

- **-Verilog**
  A switch that specifies to write the patterns in the Verilog format. The Verilog format contains pattern information and timing information from the timing file as a sequence of events. You can use the Verilog format patterns to interface to the Verilog-XL and Verifault simulators. See the “Test Pattern Formatting and Timing” chapter in the *Scan and ATPG Process Guide* for the description of the timing file and how to create it.

- **-VHdl**
  A switch that specifies to write the patterns in the VHDL format.

- **-Mode Lsi**
  A switch that writes the patterns in the LSI Verilog and WGL pattern format. You can use this mode when the first initialization vector in a Verilog or WGL pattern file is consistent with the first initialization vector in a currently supported LSITDL format file.
Save Patterns

Command Dictionary

- `timing_filename`
  An optional string that specifies the name of the file from which you want to read the non-scan event timing information.

**Examples**

The following example saves BIST patterns in Verilog format:

```plaintext
<bist circuitry setup ...>
set pattern save on
set system mode dft
<set up for random pattern simulation...>
run
report lfsrs
report statistics
set system mode bist
save patterns bist_patterns.dfta -bist_test -verilog
```

**Related Commands**

- Set Pattern Save
Save Schematic

Tools Supported: DFTAdvisor and DFTInsight
Scope: All modes

Usage

SAVE SChematic \textit{filename} [-Replace]

DFTInsight Menu Path:
\begin{itemize}
  \item File > Save > Schematic
\end{itemize}

Description

Saves the schematic currently displayed by DFTInsight.

The Save Schematic command saves the netlist currently viewed in DFTInsight for later examination. In order to view a previously saved schematic, execute the File > Open > Schematic menu item.

Arguments

- \textit{filename}
  
  A required string that specifies the name of the schematic file.

- -Replace
  
  An optional switch that specifies replacement of the contents of \textit{filename}, if a file by that name already exists.

Examples

The following example invokes the schematic viewer, creates and displays a netlist, saves the netlist, and then terminates the viewing session:

\begin{verbatim}
open schematic viewer
analyze drc violation c2-1
save schematic mydesign.v -replace
close schematic viewer
\end{verbatim}

Related Commands

- Close Schematic Viewer
- Open Schematic Viewer
- Set Schematic Display
Select Object

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

Usage

SELect OBject -ALl | {{gate_id# | pin_pathname | instance_name}... [-ADd]}

DFTInsight Menu Path:
Display > Selection > Select All

Description

Selects the specified objects in the DFTInsight schematic view.

The Select Object command selects either all the objects in the schematic view or only the objects that you specify. You can make the selection additive, that is, add the objects that you specify to those already selected, by using the -Add switch.

Arguments

- **-ALl**
  
  A switch that selects all the gates in the design.

- **gate_id#**
  
  A repeatable integer that specifies the gate identification number of the objects to select. The value of the gate_id# argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- **pin_pathname**
  
  A repeatable string that specifies the name of a pin whose gate you want to select.

- **instance_name**
  
  A repeatable string that specifies the name of the instance to select.
• -ADd

An optional switch that adds the objects that you specify to the selection list.

**Examples**

The following example selects one object and then adds two more objects to the selection list:

```
select object /i$144/q
select object /i$142/q /i$141/q
```

Now all three objects are in the selection list.
Set Bist Initialization

Scope: All modes

Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

SET Bist Initialization { 0 | 1 | No_init }

Description

Specifies the scan chains input value which indicates the states of the scan cells before FastScan applies BIST patterns.

The Set Bist Initialization command defines the initial load values on the scan chains. FastScan loads the scan chains using this input value during the simulation of BIST patterns. Then, just prior to the first BIST pattern, FastScan unloads the scan cell values into the MISR.

Arguments

- 0 | 1
  A literal that specifies to load the scan chains with all 0 or 1 values which get unloaded into the MISR.

- No_init
  A literal that specifies not to initialize the scan chains since an AND gate is used at the end of each scan chain which zeros-out the scan chain values as they are unloaded into the MISR.

Examples

The following example specifies to load the scan chains with all 1 values:

  set bist initialization 1

Related Commands

Add Mtpi Controller
Add Mtpi Output
Delete Mtpi Controller
Delete Mtpi Output

Report Mtpi Controller
Save Patterns
Set Pattern Save
Set Capture Clock

Scope: All modes

Usage
SET CApture Clock {primary_input_pin | clock_procedure_name} [-Atpg]

Description
Specifies the capture clock name for random pattern simulation.

The Set Capture Clock command specifies the name of the capture clock that the tool uses during random pattern simulation. You can specify the name of either a specific pin or a clock procedure in a test procedure file that identifies the pin. In either case, the pin must be a currently defined clock pin. Also, the capture clock that you specify cannot have a pin constraint.

If you do not specify a capture clock with this command, DFTAdvisor sets the capture clock to none. If there is no capture clock and there is only one clock in the circuit that is not a set or reset line, DFTAdvisor sets that clock as the capture clock during the rules checking and displays a warning message identifying the capture clock.

You can use the Report Environment command to list the capture clock and the Report Clocks command to identify the current list of clocks.

You use this command primarily for implementing Built-In Self-Test (BIST) circuitry.

Arguments
- primary_input_pin
  A string that specifies the name of the primary input pin that you want to assign as the capture clock.

- clock_procedure_name
  A string that specifies the name of the clock procedure in the test procedure file that identifies the primary input pin that you want to assign as the capture clock.
- **-Atpg**

  An optional switch that specifies for the tool to use the capture clock for all scan patterns it creates during the ATPG process and places in the internal pattern set.

  If you specify a `clock_procedure_name` with the -Atpg switch, then, in the test procedure file, you can apply the clock procedure to every clock cycle.

**Examples**

The following example specifies a capture clock:

```
add clocks 1 clock1
set capture clock clock1
set system mode dft
```

**Related Commands**

- Add Clocks
- Delete Clocks
- Report Clocks
- Report Environment
Set Contention Check

Scope: All modes

Usage

SET CONTENTION CHECK OFF | {ON [-ATpg] [-Start frame#]} [-Bus | -Port | -ALl]

Description

Specifies whether DFTAdvisor checks the gate types that you determine for contention.

The Set Contention Check command specifies whether contention checking is on and the conditions under which the tool performs the checks. When contention checking is on, DFTAdvisor checks for contention during the ATPG-based identification run. Contention checking is set to On upon invocation of the tool.

Arguments

- **OFF | ON**
  A literal that specifies whether the tool should perform contention checking during simulation without propagating captured data effects. The invocation default behavior is On.

- **-ATpg**
  An optional switch that specifies for DFTAdvisor to use deterministic fault simulation when identifying non-scan cells.

- **-Start frame#**
  An optional switch and integer pair that specifies the number of time frames after design initialization when DFTAdvisor begins the contention check. The default is time frame 0.

  Due to sequential initialization, the initial states on a bus may be unknown and possible contention may be unavoidable. Thus, this switch allows you to begin the contention checking after the design has initialized.

- **-Bus**
  An optional switch that specifies for DFTAdvisor to perform contention checking tri-state driver buses. This is the default.
Tri-state logic allows several bus drivers to time share a bus. However, if the circuit enables two bus drivers of opposite logic to drive the bus, physical damage can happen. This switch allows the tool to identify these conditions and notify you of their existence.

- **-Port**
  An optional switch that specifies for DFTAdvisor to perform contention checking for multiple-port flip-flops and latches. The tool identifies any multiple-port latch or flip-flop that has more than one clock, set, or reset input active (or at X).

- **-All**
  An optional switch that specifies for DFTAdvisor to perform contention checking for both tri-state driver buses and multiple-port flip-flops and latches.

**Examples**

The following example performs contention checking on both multiple-port sequential gates and tri-state busses, stops the simulation if any bus contention occurs, and displays an error message which will indicate the gate on which the contention occurred:

```plaintext
set system mode dft
set contention check on -all
run
```
Set Control Threshold

Scope: All modes

Usage

SET COntrl Threshold integer

Description

Specifies the controllability value for simulation-based pseudorandom random pattern test point identification.

The Set Control Threshold command specifies the minimum number of times a gate must be at a zero and a one state during simulation. The simulation is a set of pseudorandom random patterns defined for test point insertion using the Setup Test_point Identification command. Gates that do not meet the threshold are candidates for test points. You use this command primarily for Built-In Self-Test (BIST) circuitry.

Arguments

• integer

A required integer, greater than or equal to 0, that specifies the controllability value. The default upon DFTAdvisor invocation is 4.

Examples

The following example sets the threshold number to determine the controllability effects during pseudorandom pattern test point identification:

```plaintext
set control threshold 2
set observe threshold 4
setup test_point identification -control 9 -obs 20 -patterns 32000
  -cshare 16 -oshare 16 -base simulation
set system mode dft
setup scan identification none
run
```

Related Commands

- Set Observe Threshold
- Set Random Patterns
- Setup Scan Identification
- Setup Test_point Identification
Set Dofile Abort

Scope: All modes

Usage

SET DOfile Abort ON | OFF

Description

Lets you specify that the tool complete processing of all commands in a dofile regardless of an error detection.

By default, if an error occurs during the execution of a dofile, processing stops, and the line number of the error in the dofile is reported. The Set Dofile Abort command lets you turn this functionality off so that the tool continues to process all commands in the dofile.

Arguments

- **ON**
  
  A required literal that halts the execution of a dofile upon the detection of an error. This is the default upon invocation of the tool.

- **OFF**

  A required literal that forces dofile processing to complete all commands in a dofile regardless of error detection.

Examples

The following example sets the Set Dofile Abort command off to ensure that all commands in `test1.dofile` are executed.

```
set system mode dft
set dofile abort off
dofile test1.dofile
```

Related Commands

Dofile
Set Drc Handling

Scope: Setup and Dft modes

Usage

SET DRc Handling rule_id [Error | Warning | NOTe | Ignore] [NOVerbose | Verbose] [NOAtpg_analysis | Atpg_analysis] [-Mode {Sequential | Combinational}]

Description

Specifies how DFTAdvisor globally handles design rule violations.

The Set Drc Handling command specifies the handling of the messages for the scan cell RAM rules checking, Clock rules checking, Data rules checking, Extra rules checking, procedure file checking, and Trace rules checking. You can specify that the violation messages for these checks be either error, warning, note, or ignore.

The Set Drc Handling command does not support the scannability (S) rules.

Each rules violation has an associated occurrence message and summary message. The tool displays the occurrence message only for either error conditions or if you specify the Verbose option for that rule. The tool displays the rule identification number in all rules violation messages.

The Atpg_analysis option provides full test generation analysis when performing rules checking for some clock (C) rules, for some data (D) rules, and for some extra (E) rules. For example, if you specify Atpg_analysis for clock rule C1 and the tool simulates a clock input to be X, the rule violation occurs when it is possible for the test generator to create a test pattern while that clock input is on, all defined clocks are set off, and constrained pins are set to their constrained state.

When you specify Atpg_analysis, the tool requires some additional CPU time and memory to perform the full test generation analysis.
Arguments

• `rule_id`

A required literal that specifies the identification of the exact design rule violations whose message handling you want to change.

The design rule violations and their identification literals are divided into the following six groups: RAM, Clock, Data, Extra, Procedure, and Trace rules violation IDs.

The following lists the RAM rules violation IDs. For a complete description of these violations refer to the “RAM Rules” section in the Design-for-Test Common Resources Manual.

A1 — When all write control lines are at their off-state, all write, set, and reset inputs of RAMS must be at their inactive state.

A2 — A defined scan clock must not propagate to a RAM gate, except for its read lines.

A3 — A write or read control line must not propagate to an address line of a RAM gate.

A4 — A write or read control line must not propagate to a data line of a RAM gate.

A5 — A RAM gate must not propagate to another RAM gate.

A6 — All the write inputs of all RAMs and all read inputs of all data_hold RAMs must be at their off-state during all test procedures, except test_setup.

A7 — When all read control lines are at their off-state, all read inputs of RAMs with the read_off attribute set to hold must be at their inactive state.

The following lists the BIST rule violation IDs. For a complete description of these violations refer to the “BIST Rules” section in the Design-for-Test Common Resources Manual.

B2 — Every scan chain input pin must connect to an LFSR.

The following lists the Clock rules violation IDs. For a complete description of these violations refer to the “Clock Rules” section in the Design-for-Test Common Resources Manual.
C1 — The netlist contains the unstable sequential element in addition to the backtrace cone for each of its clock inputs. The pin data shows the value that the tool simulates when all the clocks are at their off-states and when the tool has set all the pin constraints to their constrained values.

C2 — The netlist contains the failing clock pin and the gates in the path from it to the nearest sequential element (or primary input if there is no sequential element in the path.) The pin data shows the value that the tool simulates when the failing clock is set to X, all other clocks are at their off-states, and when the tool has set all pin constrains to their constrained values.

C3 | C4 — The netlist contains all gates between the source cell and the failing cell, the failing clock and the failing cell, and the failing clock and the source cell. The pin data shows the clock cone data for the failing clock.

C5 | C6 — The netlist contains all gates between the failing clock and the failing cell. The pin data shows the clock cone data for the failing clock.

C7 — The netlist contains all the gates in the backtrace cone of the bad clock input of the failing cell. The pin data shows the constrained values.

C8 | C9 — The netlist contains all the gates in the backtrace cone of the failing primary output. The pin data shows the clock cone data for the failing clock.

The following lists the Data rules violation IDs. For a complete description of these violations refer to the “Scan Cell Data Rules” section in the Design-for-Test Common Resources Manual.

D1 — The netlist contains all the gates in the backtrace cone of the clock inputs of the disturbed scan cell. The pin data shows the pattern values the tool simulated when it encountered the error.

D2 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulated for all time periods of the shift procedure.

D3 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the master_observe procedure.
D4 — The netlist contains all the gates in the backtrace cone of the failing gate. The pin data shows the values the tool simulates for all time periods of the \texttt{skew\_load} procedure.

D5 — The netlist contains the disturbed gate, and there is no pin data.

D6 | D7 | D8 — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the value that the tool simulates when all clocks are at their off-states.

D9 — The netlist contains all the gates in the backtrace cone of the clock inputs of the failing gate. The pin data shows the pattern value the tool simulated when it encountered the error.

The following lists the Extra rules violation IDs. For a complete description of these violations refer to the “Extra Rules” section in the Design-for-Test Common Resources Manual.

E1 — All scan cells must be LSSD scan cells that contain a master and slave latch.

E2 — There must be no inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell.

E3 — There must be no inversion between MASTER and SLAVE for any scan cell.

E4 — Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures.

E5 — When constrained pins are placed at their constrained states, and binary states are placed on PIs and scan cells, X states must no be capable of propagating to an observable point.

E6 — When constrained pins are placed at their constrained states, the inputs of a gate must not have sensitzizable connectivity to more than one memory element of a scan cell.

E7 — External bidirectional drivers must be at the high impedance (Z) state during the application of the test procedure.

E8 — All masters of all scan cells of a scan chain must use a single shift clock.
**E9** — The drivers of wire gates must not be capable of driving opposing binary values.

**E10** — Performs bus contention mutual-exclusivity checking. Similar to E4, but does not check for this condition during test procedures.

**E11** — The ability of a bus gate to attain a Z state.

**E12** — The test procedures cannot violate any ATPG constraints.

**E13** — Satisfy both ATPG constraints and bus contention prevention (for buses that fail rule E10)

The following lists the Procedure rules violation IDs for which you can change their handling to “ignore”. Otherwise these are errors. For a complete description of these violations refer to the “Procedure Rules” section in the *Design-for-Test Common Resources Manual*:

**P30** — A procedure may not place a clock at its on-state at the same time it forces a non-clock pin to a value or place another clock at its off-state.

**P31** — A procedure may not force a non-clock pin to a value at the same time it forces a clock pin to a value.

**P32** — A procedure may not place a clock at its off-state at the same time it places another clock at its on-state.

**P33** — When a pattern places a clock at its off-state, all clocks must be at their off-state.

The following lists the Trace rules violation IDs. For a complete description of these violations refer to the “Scan Chain Trace Rules” section in the *Design-for-Test Common Resources Manual*:

**T1** — All defined scan chains must contain at least on scan cell

**T2** — The netlist contains the blocked gate. The pin data shows the values the tool simulates for all time periods of the *shift* procedure.

**T3** — The netlist contains all the gates in the backtrace cone of the blocked gate. The pin data shows the values the tool simulates for all time periods of the *shift* procedure.
T4 — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T5 | T6 — The netlist contains all the gates in the backtrace cone of the clock inputs of the blocked gate. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T7 — The netlist contains all the gates in the path between the two failing latches. The pin data shows the values the tool simulates for all time periods of the shift procedure.

T9 — The traced scan chain input pin must be the same as the entered scan chain input pin.

T11 — A clock input of the memory element closest to the scan chain input must not be turned on during the shift procedure prior to the time of the force_sci statement.

T12 — If a scan cell contains a SLAVE, the MASTER is not directly observable.

T16 — When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink gate (PGS) in the long path must be at the non-controlling value of the PGS gate.

T17 — Reconvergent pulse generator sink gates cannot be connected to any of the following: primary outputs, non-clock inputs of the scan memory elements, ROM gates, non-write inputs of RAMs and transparent latches.

T18 — The maximum traced number of cells in the longest scan chain of a group must equal the entered number of repetitions in the apply shift statement in the load_unload procedure.

T19 — If a scan cell has a SLAVE, then all scan cells must have a SLAVE.

T20 — The number of shifts specified using the Set Number Shifts command must be at least equal to the length of the longest scan chain,

- Error

An optional literal that specifies for the tool to both display the error occurrence message and immediately terminate the rules checking.
• **Warning**

An optional literal that specifies for the tool to display the warning summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

• **NOTe**

An optional literal that specifies for the tool to display the summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

• **Ignore**

An optional literal that specifies for the tool to not display any message for the rule’s violations. The tool must still some rules and they must pass to allow certain functions to be performed later.

• **NOVerbose**

An optional literal that specifies for the tool to display the occurrence message only once for the rules violation. This is the default.

• **Verbose**

An optional literal that specifies for the tool to display the occurrence message for each occurrence of the rules violation.

• **NOAtpg_analysis**

An optional literal that specifies for the tool to not use full test generation analysis when performing rules checking. This is the default.

• **Atpg_analysis**

An optional literal that specifies for the tool to use full test generation analysis when performing rules checking for clock rules (like C1, C3, C4, and C5), some D rules (like D6 and D9), and some E rules (like E4, E5, E8, E10, E11, and E12).

Note: If you want the tool to use the constraint values during the D6 rule analysis, you need to use the Atpg_analysis option.
Set Drc Handling

-Mode {Combinational | Sequential}

An optional switch and literal for the tool to use with the E10 rule. The Combination option is the default upon invocation of DFTAdvisor. It performs bus contention mutual-exclusivity checking. This checking differs from rule E4 in that it does not check for this condition during test procedures.

The Sequential option considers the inputs to a single level of sequential cells behaving as “staging” latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enables of the tri-state devices except through the sequential cells.

Examples

The following example specifies rule checking E4 to be an error:

```
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 outdata4
add clocks 1 clock1
add clocks 0 clock2
set drc handling e4 error
set system mode dft
```

Related Commands

Report Drc Rules                     Set Flatten Handling
Set Fault Sampling

Scope: All modes

Usage

SET FAult Sampling percentage

Description

Specifies the fault sampling percentage for scan identification.

The Set Fault Sampling command specifies the fault sampling percentage that DFTAdvisor uses for scan identification. The default upon invocation of the tool is to use all faults (100%) in the internally generated fault list for scan identification.

Fault sampling allows you to process a fraction of the total faults and thus decrease process time when you need to evaluate a large circuit. Once you specify a percentage, the tool randomly picks the sample to process.

Arguments

- **percentage**
  A required positive integer from 1 to 100 that specifies the fault sampling percentage that you want DFTAdvisor to use for scan identification. The invocation default is 100 percent.

Examples

The following example performs scan identification with only 50% of the internally generated fault list.

```plaintext
add clocks 0 clock
set system mode dft
set fault sampling 50
run
report scan identification
```
Set Flatten Handling

Scope: All modes

Usage

SET FLatten Handling rule_id [Error | Warning | NOTe | Ignore] [Verbose | NOVerbose]

Description

Specifies how DFTAdvisor globally handles flattening violations.

The Set Flatten Handling command specifies the handling of the messages for net checking, pin checking, and gate checking. You can specify that the violation messages for these checks be either error, warning, note, or ignored.

Each rules violation has an associated occurrence message and summary message. The tool displays the occurrence message only for either error conditions or if you specify the Verbose option for that rule. The tool displays the rule identification number in all rules violation messages.

Arguments

- rule_id

  A required literal that specifies the identification of the exact flattening rule violations whose message handling you want to change.

  The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

  Following are the net rules:

  FN1 — A module net is floating. The default upon invocation is warning.

  FN2 — A module net has driver and constant value property. The default upon invocation is warning and its property is not used.

  FN3 — An instance net is floating. The default upon invocation is warning.

  FN4 — An instance net is not used. The default upon invocation is warning.

  FN5 — A multiple driven wired net. The default upon invocation is warning.
FN6 — A bus net attribute cannot be used. The default upon invocation is warning.
FN7 — Two connected nets have inconsistent net attributes. The default upon invocation is warning and both attributes are not used.
FN8 — Parallel wired behavior. The default upon invocation is warning.
FN9 — The bus net has multiple different bus keepers. The default upon invocation is warning and their effects are additive.

Following are the pin rules:
FP1 — The circuit has no primary inputs. The default upon invocation is warning.
FP2 — The circuit has no primary outputs. The default upon invocation is warning.
FP3 — The primary input drives logic gates and switch gates. The default upon invocation is warning.
FP4 — A pin is moved. The default upon invocation is warning.
FP5 — A pin was deleted by merging. The default upon invocation is warning.
FP6 — Merged wired in/out pins. The default upon invocation is warning.
FP7 — Merged wired input and output pins. The default upon invocation is warning.
FP8 — A module boundary pin has no name. The default upon invocation is warning.
FP9 — An in/out pin is used as output only. The default upon invocation is ignored.
FP10 — An output pin is used as in/out pin. The default upon invocation is ignored.
FP11 — An input pin is used as in/out pin. The default upon invocation is ignored.
FP12 — An output pin has no fan-out. The default upon invocation is ignored.
**FP13** — An input pin is floating. The default upon invocation is warning

Following are the gate rules:

**FG1** — The defining model of an instance does not exist. The default upon invocation is error. If it is not an error condition, this instance is treated as an undefined primitive.

**FG2** — The feedback gate is not in feedback loop. The default upon invocation is error.

**FG3** — The bus keeper has no functional impact. The default upon invocation is warning

**FG4** — The RAM/ROM read attribute not supported. The default upon invocation is warning

**FG5** — The RAM attribute not supported. The default upon invocation is warning

**FG6** — The RAM type not supported. The default upon invocation is error

**FG7** — The netlist module has a primitive not supported. The default upon invocation is error. If non-error is chosen, this primitive is treated as undefined.

**FG8** — The library model has a primitive not supported. The default upon invocation is error. If non-error is chosen, this primitive is treated as undefined.

- **Error**
  
  An optional literal that specifies for the tool to both display the error occurrence message and immediately terminate the rules checking.

- **Warning**
  
  An optional literal that specifies for the tool to display the warning summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

- **NOTe**
  
  An optional literal that specifies for the tool to display the summary message indicating the number of times the rule was violated. If you also specify the
Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

- **Ignore**
  An optional literal that specifies for the tool to not display any message for the rule’s violations. The tool must still check some rules and they must pass to allow certain functions to be performed later.

- **NOVerbose**
  An optional literal that specifies for the tool to display the occurrence message only once for the rules violation and give a summary of the number of violations. This is the default.

- **Verbose**
  An optional literal that specifies for the tool to display the occurrence message for each occurrence of the rules violation.

**Example**

The following example changes the handling of the FG7 flattening rule to warning and specifies that each occurrence should be listed:

```
set flatten handling fg7 warning verbose
```

**Related Commands**

- Report Flatten Rules
- Set Drc Handling
Set Gate Level

Tools Supported: DFTAdvisor and DFTInsight
Scope: All modes

Usage

SET GAte Level **Primitive | Design | Low_design**

DFTInsight Menu Path:
  Setup > Design Level > Design | Primitive

Description

Specifies the hierarchical level of gate reporting and displaying.

The Set Gate Level command specifies the hierarchical gate level at which the tool reports and schematically displays gate information. Once you set the gate level, the tool processes all subsequent report and display commands using the new gate level.

Whenever you issue a command which invalidates the flattened model, the tool also invalidates the hierarchical gate display structure. This causes DFTInsight to clear the schematic view area. You can rebuild the hierarchical gate structure by creating a new flattened model. To do so enter and exit the Setup mode.

Arguments

- **Primitive**
  A literal that specifies to display gate information at the built-in primitive gate level.

- **Design**
  A literal that specifies to display gate information at the design library hierarchical gate level. These are the top level cells of the design library which are instantiated in your design. This is the default upon invocation of the tool.

- **Low_design**
  A literal that specifies to display gate information at the pseudo-hierarchical gate level. A pseudo-hierarchical gate is a cluster gate that contains primitive
gates and is at the lowest hierarchy level in the design library. These gates only differ from design level gates if the library contains macro cells.

**Examples**

The following example sets the gate report level so that simulated values of the gate and its inputs are shown (assuming a rules checking error occurred when exiting the Setup system mode):

```plaintext
add clocks 0 clock
set system mode dft
set gate level primitive
set gate report error_pattern
report gates i_1006/o
```

**Related Commands**

- Report Gates
- Set Gate Report
Set Gate Report

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Usage

SET GAte REport {Normal | Trace | Error_pattern | T1e_value | Constrain_value | [Drc_pattern procedure_name [time | -All]]}

DFTInsight Menu Path:
    Setup > Reporting Detail

Description

Specifies the additional display information for the Report Gates command.

The Set Gate Report command controls the type of additional information that the Report Gates command displays. Each Set Gate Report option causes the Report Gates command to provide different details regarding the gates on which it reports. This command also controls the information displayed for each instance in the DFTInsight Schematic View area.

When you exit the Setup system mode, the trace and any rules-checking error pattern results will not be available with the usage of this command.

For information on the format output by the different options in this command, refer to “Report Gates” on page 2-239.

Arguments

• Normal
    A literal that specifies for the Report Gates command to display only its standard information. This is the default mode upon invocation of the tool.

• Trace
    A literal that specifies for the Report Gates command to display the simulated values of the gates for the shift patterns. Use the Trace option to determine why a scan chain was not properly sensitized during the shift procedure.
• **Error_pattern**
  A literal that specifies for the Report Gates command to display the simulated value of the gates and its inputs for the pattern at which an audit error occurred.

• **TIE_value**
  A literal that specifies for the Report Gates command to display the simulated values that result from all natural tied gates and learned constant value non-scan cells.

• **Constrain_value**
  A literal that specifies for the Report Gates command to display the simulated values that result from all natural tied gates, learned constant value non-scan cells, constrained pins, and constrained cells.

The Report Gates command displays three values which are separated by a slash (/). These values are the gate constrained value (0, 1, X, or Z), the gate forbidden values (-, 0, 1, Z, or any combination of 01Z), and the fault blockage status (- or B, where B indicates all fault effects of this gate are blocked).

• **Drc_pattern procedure_name [-All | time]**
  Two literals and an optional time triplet that specifies the name of the procedure and the time in the test procedure file that the Report Gates command uses to display a gates simulated value.

You must set the Drc_pattern prior to rules checking and you must re-execute the design rules checking process, otherwise no data (-) from this option will be available for gate reporting.

The valid options for use with Drc_pattern are as follows:

  *procedure_name* — A literal that specifies a procedure in the test procedure file that you want the Report Gates command to use when displaying the value of a gate. The valid literals for the *procedure_name* option are as follows:

    *Test_setup* — this optional procedure sets non-scan elements to the state you desire for the *load_unload* procedure.

    *Load_unload* — this required procedure describes how to load and unload data in the scan chains.
**SHIf**t — this required procedure describes how to shift data one position down the scan chain.

**SKew_load** — this optional procedure describes how to propagate the output value of the preceding scan cell into the master memory element of the current cell (without changing the slave), for all scan cells.

**SHADOW_Control** — this optional procedure describes how to load the contents of a scan cell into the associated shadow.

**Master_**observe — this procedure describes how to place the contents of a master into the output of its scan cell.

**SHADOW_Observe** — this optional procedure describes how to place the contents of a shadow into the output of its scan cell.

- **All** — An optional switch that specifies to use all times in the test procedure file. This is the default.

  **time** — An optional positive integer greater than 0 that specifies a time in the test procedure file.

**Examples**

The following example sets the gate report so that simulated values of the gate and its inputs are shown (assuming a rules checking error occurred when exiting the setup system mode):

```
set gate report trace
set system mode dft
report gates I_1006/O
```

**Related Commands**

Report Gates       Set Gate Level
Set Identification Model

Scope: Setup mode

Usage

SET IDentification Model [-Clock {Original | None | Extra}] [-Disturb {ON | OFF}]

Description

Specifies the simulation model that DFTAdvisor uses to imitate the scan operation during the scan identification process.

The Set Identification Model command specifies a simulation model needed to imitate scan operation during scan identification; before real scan chains are inserted.

In normal scan operation, after DFTAdvisor loads values into the scan cells, a scan memory element should not change its scan-in value until the loaded value is used. To achieve this scan cell stability, all the clocks of a scan cell should be at their off-state. However, because the implementation of scan chains is not done yet, DFTAdvisor does not yet know its ability to control all the clocks. Therefore, the Set Identification Model command allows you three -Clock options, along with the -Disturb argument, to specify how DFTAdvisor is to handle scan operation during the scan identification process.

The scan identification process is performed with the Run command and this is the time when DFTAdvisor selects the non-scan cells that are to be replaced with the corresponding scan cell. If you want to display the current settings for the scan identification model, you can use the Report Environment command.

During the scan identification process, DFTAdvisor automatically decides which nonscan cells require extra logic to fix their clock behavior. When DFTAdvisor performs the scan identification, it processes each nonscan cell separately. By default, the scan identification model specifies for DFTAdvisor to use the -Clock Original option for nonscan cells that do not require the extra logic to control its clocks, and the -Clock Extra option for the non-scan cells that do require the extra logic for clock controllability.
Arguments

• -Clock Original | None | Extra

An optional switch and literal pair that determines the effect of the clocks on scan memory elements values. The three -Clock options are as follows:

Original — A literal specifying that scan memory elements operate as their current clock configuration. This is the default for nonscan cells that DFTAdvisor determines do not require extra logic for controllability of that nonscan cell’s clocks. If you specify this option, then DFTAdvisor does not add any extra logic and uses the original clock configuration for each nonscan cell on a global design-wide basis.

None — A literal specifying that after scan loading, scan memory elements can hold their value for one time frame, regardless of their clock values.

Extra — A literal specifying that external controllable clocks replace the original clocks so that the scan cells are capable of holding their scan values right after scan loading. This option is the default for nonscan cells that DFTAdvisor determines do require extra logic for controllability of that nonscan cell’s clocks. If you specify this option, then DFTAdvisor adds extra logic to every nonscan cell on a global design-wide basis.

• -Disturb ON | OFF

An optional switch and literal pair that determines the effect of scan loading on non-scan memory elements. The two -Disturb options are as follows.

ON — A literal specifying that the value of the non-scan memory elements can be disturbed by scan loading operations. This is the default. If the disturb option is on, DFTAdvisor sets the states of non-scan memory elements to the unknown (X) state after the scan loading operation.

OFF — A literal specifying that the value of non-scan memory elements cannot be disturbed by scan loading. When the disturb option is off, the states of the non-scan memory elements are the same as before the scan loading operation.
Examples

The following example forces DFTAdvisor to add extra primary input pins to replace the original clocks on a global design-wide basis:

```
  set identification model -clock extra
  set system mode dft
  setup scan identification full_scan
  run
```

Related Commands

Report Environment
Set Instancename Visibility

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke DFTInsight and have it displaying instances to see the effects of this command.

Usage

SET INstancename Visibility [ON | Off] [-Full | {[-Leaf leaf_levels]
[-Root root_levels]}

DFTInsight Menu Path:
Setup > Preferences: Instance Names

Description

Specifies whether DFTInsight displays instance names immediately above each instance in the Schematic View area.

The Set Instancename Visibility command also allows you to control the length of instance pathnames displayed in the schematic window. By using the -Full, -Leaf, and -Root options, you can control the number of hierarchical name elements displayed for each instance name.

You may specify the -Leaf and the -Root options together. If the total number of levels specified meets or exceeds the number of levels present in a name, then the entire name is displayed.

When truncation is done, the “...” characters are displayed in place of the omitted name elements.

Arguments

- ON | Off
  A literal specifying to display or not display the instance name labels. The default upon invocation is On.

- -Full
  An optional literal that specifies that instance names should contain the full hierarchical pathname. This is the default upon invocation.
-Leaf \textit{leaf\_levels}

An optional switch that specifies how many levels of a hierarchical name are displayed starting from the leaf name and counting up the hierarchy. The value of \textit{leaf\_levels} must be greater than 0 in order for the -Leaf option to be valid. If the -Leaf option is specified with \textit{leaf\_levels} omitted, the default value will be set to 1 and only the leaf name will be shown(.../leafname). Otherwise a maximum of \textit{leaf\_levels} of name elements is shown.

-Root \textit{root\_levels}

An optional switch that specifies how many levels of a hierarchical name are displayed starting from the root and counting down the hierarchy. The value of \textit{root\_levels} must be greater than 0 in order for the -Root option to be valid. If the -Root option is specified with \textit{root\_levels} omitted, the default value is set to 1 and only the root name will be shown. Otherwise a maximum of \textit{root\_levels} of name elements is shown.

\textbf{Examples}

Given an instance name “top/alu/add1/u3,”

1. Specifying the following:

\texttt{set instancename visibility -r 1 -l}

Results in the following display:

/\textit{top}/*/\textit{...}/u3

2. Specifying the following:

\texttt{set instancename visibility -r 3 -l 3}

Results in the following display:

/\textit{top}/\textit{alu}/add1/u3

3. Specifying the following:

\texttt{set instancename visibility -l 2}

Results in the following display:

*/\textit{add1}/u3
Set Internal Fault

Scope: Setup mode

Usage

SET INternal Fault ON | OFF

Description

Specifies whether the tool allows faults within or on the boundary of library models.

The Set Internal Fault command specifies whether the tool will allow faults on internal nodes of library models or only on the library model boundary. The default upon invocation of the tool is to allow faults on the internal nodes of library models.

Arguments

- **ON**
  A literal that specifies to allow faults on the internal nodes of library models. This is the default upon invocation of the tool.

- **OFF**
  A literal that specifies to allow faults only on the boundary of the library models.

Examples

The following example invokes the scan identification process such that DFTAdvisor does not take into consideration any internal faults when in calculating the efficiency percentage:

```
set internal fault off
set system mode dft
setup scan identification full_scan
run
```
Set Io Insertion

Scope: All modes

Prerequisites: Input and output buffers must be defined in either the ATPG library or using the Add Cell Models command.

Usage

SET IO Insertion ON | OFF | { [TEn] [Ram] [SEn] [TCIks] [SIns] [SOuts] [Control] [OBserve] [-Model model_name] }

Description

Specifies whether to insert I/O buffers.

The Set IO Insertion command specifies whether DFTAdvisor should insert I/O buffers automatically during scan insertion. By having automatic I/O buffer insertion turned off (the default), you can perform scan insertion at the block level or insert the I/O buffers manually after inserting scan at the design level.

If you defined I/O buffers in the ATPG library or used the Add Cell Models command to define them, when you set this command to “on”, DFTAdvisor will automatically insert the I/O buffers during scan insertion.

You can specify which test control signals should have I/O buffers added. You can specify one or more of the test signal literal arguments. The specified signals can be internal signals (output port of a library cell) or new pins generated by DFTAdvisor.

The Set IO Insertion command is additive. This means that each time you issue the command, it adds any new options to those already defined.

Arguments

- **ON | OFF**

  A required literal that specifies whether to insert I/O buffers for all test signals. If you are not turning On or Off all test signals, you must specify at least one of the test signal arguments. If you want to remove any existing I/O buffer signals from the list of signals to buffer, you turn off I/O buffer insertion (Set IO Insertion off). The default upon invocation is off.
• **TEn**
  A literal that specifies to buffer the test_enable pin.

• **Ram**
  A literal that specifies to buffer the ram_write_control and ram_read_control pins.

• **SEn**
  A literal that specifies to buffer the scan_enable pin(s).

• **TCIks**
  A literal that specifies to buffer all test clock pins, including clock, set, and reset.

• **SIns**
  A literal that specifies to buffer all scan_in pins of inserted scan chains.

• **SOuts**
  A literal that specifies to buffer all scan_out pins of inserted scan chains.

• **Control**
  A literal that specifies to buffer all test point control pins, if no scan cell is requested with the Setup Test_point Insertion command.

• **OBserve**
  A literal that specifies to buffer all test point observe pins, if no scan cell is requested with the Setup Test_point Insertion command.

• **-Model** *model_name*
  An optional switch and string pair that specifies the name of a buffer in the ATPG library for DFTAdvisor to insert on the test pins. You must first identify the buffer with either the Add Cell Models command or with the cell_type library attribute. The specified model should be the OUTBUF type for scan outputs and the INBUF type for all scan inputs and test signals.

  If you do not use the -Model switch, by default, DFTAdvisor uses the first buffer model in the buffer cell model list (which you can see with the Report Cell Models command).
Examples

The following example shows how to enable the adding of I/O buffers automatically to all test control signals:

    set io insertion on

To enable the adding of I/O buffers to only the scan in, scan out, control, and observe signals, enter:

    set io insertion sins souts control observe

Related Commands

Add Buffer Insertion  Add Cell Models
Set Latch Handling

Scope: Setup mode

Usage

SET LAtch Handling None | Scan

Description

Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.

The Set Latch Handling command specifies whether the tool can consider non-transparent latches as candidates for scan insertion. If you determine that the tool is to consider non-transparent latches as candidates for scan insertion, you must turn on the appropriate Set Test Logic command settings before DFTAdvisor performs the rules checking process.

If you use the Set Drc Handling command to set the D6 rule to error or warning, D6 checks the non-scannable latches for transparency. By default, if they are not transparent and you turned test logic insertion on, DFTAdvisor does not consider the non-transparent latches for scan insertion. However, if you use the Scan argument with the Set Latch Handling command, DFTAdvisor will consider the non-transparent latches for scan insertion.

If you use the Set Drc Handling command to ignore the D6 rule, then rules checking does not check the non-scannable latches for transparency and DFTAdvisor will automatically consider all non-scannable latches, whose test logic you turned on, for scan insertion.

Arguments

- None
  A literal specifying to give no consideration to non-transparent latches for scan insertion. This is the default upon invocation of DFTAdvisor.

- Scan
  A literal specifying to consider non-transparent latches for scan insertion when test logic is turned on.
Related Commands

Set Drc Handling          Set Test Logic
Set Lockup Latch

Scope: Dft mode

Prerequisites: You must first specify the latch model that you want DFTAdvisor to use with the Add Cell Models command.

Usage


Description

 Specifies for DFTAdvisor to insert latches between different clock domains to synchronize the clocks within a scan chain.

 If you specify for DFTAdvisor to merge scan cells with different shift clocks into the same scan chain with the Insert Test Logic command, then you can synchronize the clock edges by inserting latches between the scan cells that are in different clock domains.

 Before you can issue the Set Lockup Latch command, you must identify the latch you want DFTAdvisor to use with the Add Cell Model -Type DLAT command.

 For more information on grouping clocks and inserting lockup latches, refer to “Merging Chains with Different Shift Clocks” in the Scan and ATPG Process Guide.

 If you define any lockup latches in the cell order file, this command is not used. If no lockup latches are defined in that file, DFTAdvisor uses the settings in this command. For more information on the cell order file, refer to the Insert Scan Chains and Insert Test Logic commands.

Arguments

- **ON | OFF**
  A literal that specifies whether DFTAdvisor should insert latches between different clock groups within a scan chain. The default behavior upon invocation is OFF.
• **-Nolast | -Last**
  An optional switch that specifies whether DFTAdvisor should insert a latch between the last scan cell and the scan out pin of a scan chain. The default behavior upon invocation of DFTAdvisor is -Nolast.

• **-First_clock | -SEcond_clock**
  An optional switch that specifies for DFTAdvisor to connect either the clock of the first set of scan cells or the clock of the second set of scan cells to the clock input of the lockup latch. The default behavior upon invocation of DFTAdvisor is -First_clock.

• **-STABLE_High latch_model1**
  An optional switch and string that specify that the latch model is to be used as a lockup latch in between stable high clock domains. If you do not specify this option, any defined Dlat model is used. Before using this option, you must define the latch_model1 using the Add Cell Models command with the -Type Dlat option.

• **-STABLE_Low latch_model2**
  An optional switch and string that specify that the latch model is to be used as a lockup latch in between stable low clock domains. If you do not specify this option, any defined Dlat model is used. Before using this option, you must define the latch_model2 using the Add Cell Models command with the -Type Dlat option.

• **-Internal | -NOInternal**
  An optional switch that specifies for DFTAdvisor to insert lockup latches between different internally generated clocks. This is often used with subchains that have internal clocks that are not driven. A clock must not have a path to a primary input clock for it to be considered an internal clock. This includes clocks that are accessible at the primary input clock through test logic (muxes) or sensitized gates. The default behavior upon invocation of DFTAdvisor is -Nointernal.

**Examples**

The following example defines two different groups of clocks, identifies the latch that DFTAdvisor is to insert if the clocks need synchronization, enables lockup latch insertion, and performs the insertions. The -Clock Merge option informs
DFTAdvisor to combine the scan cells that are associated with the clocks (that are in the same clock group) together into a single scan chain.

```
add clocks 0 clk1 clk2 clk3
add clocks 1 clk4 clk5 clk6
add clock groups group1 clk1 clk2 clk3
add clock groups group2 clk4 clk5 clk6
add cell model dlat1a -type dlat enable data
set lockup latch on
insert test logic -scan on -clock merge
```

DFTAdvisor inserts lockup latches between clk1 and clk2, clk2 and clk3, clk4 and clk5, and clk5 and clk6. Insert Test Logic is the command that actually causes DFTAdvisor to substitute the identified non-scan cells with the scan replacements and also place the lockup latches into the design.

![Note]

The previous example causes DFTAdvisor to create two scan chains because there are two clock groups.

**Related Commands**

- Add Cell Models
- Add Clock Groups
- Add Test Points
- Delete Test Points
- Insert Test Logic
- Report Test Points
Set Logfile Handling

Scope: All modes

Usage

SET LOGfile Handling {[filename] [-Replace | -Append]}

Description

Specifies for DFTAdvisor to direct the transcript information to a file.

The Set Logfile Handling command causes DFTAdvisor to write the transcript information, which includes the commands and the corresponding output (if any), into the file you specify. You can execute the Set Logfile Handling command at any time within DFTA, and you can also execute it multiple times.

In the logfile, all commands that DFTAdvisor executes are preceded with the command keyword. You can easily search for the commands you executed, and then you can generate a separate dofile containing those commands which you can rerun within DFTAdvisor.

When you set the logfile handling, DFTAdvisor still writes the same information to the session transcript window in addition to the logfile. However, you can disable the writing of the information to the transcript window with the Set Screen Display command.

If you want DFTAdvisor to stop writing to a logfile, issue the Set Logfile Handling command with no options, which closes the appropriate files.

Arguments

- *filename*
  
  An optional string that specifies the name of the file where you want DFTAdvisor to write the transcript output. This string can be a full pathname or a leafname. If you only specify a leafname, the tool creates the file in the directory from which you invoked the tool.

- -Replace
  
  An optional switch that forces DFTAdvisor to overwrite the file if a file by that name already exists.
- Append

An optional switch that causes DFTAdvisor to begin writing the transcript at the end of the specified file.

Examples

The following example specifies for DFTAdvisor to write a logfile and to disable the writing of the transcript:

```plaintext
set logfile handling /user/designs/setup_logfile
set screen display off
add clocks 0 clk
add clocks 1 pre clr
report clocks
```

The following information shows what the logfile contains after running the preceding set of commands:

```plaintext
// command: set scr d off
// command: add clocks 0 clk
// command: add clocks 1 pre clr
// command: report clocks
PRE, off_state 1
CLR, off_state 1
CLK, off_state 0
```

Related Commands

Set Screen Display
Set Loop Duplication

Scope: All modes

Prerequisites: You must use this command before the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode.

Usage

SET Looop Duplication [ON | OFf]

Description

Specifies whether to include duplicate gates in feedback paths which are generated during the circuit flattening process.

The Set Loop Duplication command determines whether the flattening process generates duplicate gates within an identified feedback path. By default, the feedback paths include any duplicated gates. You suppress the duplicated gates by turning the loop duplication off prior to initiating the circuit flattening process.

You use this command in conjunction with the Report Feedback Paths and Add Display Loop commands.

Arguments

- **ON**
  
  An optional literal that specifies for the circuit flattening process to generate duplicate gates within any identified feedback paths. This is the default.

- **OFf**
  
  An optional literal that specifies for the circuit flattening process to not generate duplicate gates within any identified feedback paths.
Examples

The following example invokes the optional schematic viewing application, turns off loop duplication, leaves the Setup mode which among other things flattens the simulation model and performs the learning process, displays the identification numbers of any learned feedback paths, and then schematically displays one of the feedback paths without any gate duplication:

```
open schematic viewer
set loop duplication off
set system mode dft
report feedback paths
Loop#=0, feedback_buffer=26, #gates_in_network=5
INV /I_956__I_582/ (51)
PBUS /I_956__I_582/N1/ (96)
ZVAL /I_956__I_582/N1/ (101)
INV /I_956__I_582/ (106)
TIEX /I_956__I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
INV /I_962__I_582/ (52)
PBUS /I_962__I_582/N1/ (95)
ZVAL /I_962__I_582/N1/ (100)
INV /I_962__I_582/ (105)
TIEX /I_962__I_582/ (27)
```

add display loop 1

Related Commands

- Add Display Loop
- Report Feedback Paths
**Set Multiple Scan_enables**

Scope: Setup and DFT modes

Prerequisites: You must be using mux-DFF scan type to enable this command.

**Usage**

SET MUltiple Scan_enables {OFF | ON}

**Description**

Specifies to create multiple scan_enables.

The Set Multiple Scan_enable command specifies to create an independent scan_enable for each scan chain associated with each clock edge of each clock domain. This assumes that an independent scan chain is created for each clock edge in each clock domain. The invocation default is to create one scan enable to control all scan chains.

This command is useful for at-speed scan test where a dedicated scan enable may be required for each clock domain.

Note

You can only use multiple scan enable signals with mux-DFF scan type. Also, all multiple scan enable signals must be external pins.

For example, if you turn on multiple scan enables and clock A has scan elements of both rising and falling edges, then two scan chains are created. The first scan chain contains the rising edge scan elements, while the second contains the falling edge scan elements. DFTAdvisor then creates a separate scan enable for each of these two scan chains (two scan enables).

You can control the number of scan chains created using the Insert Scan Chains or Insert Test Logic commands with the “-Clock {Nomerge | Merge}” or “-Edge {Nomerge | Merge}” options.

The name of the scan enable signals is determined by the Setup Scan Insertion command with the -SEN option.
Arguments

- **OFF**
  A literal that specifies DFTAdvisor to create only one scan enable to control all scan chains. This is the invocation default.

- **ON**
  A literal that specifies DFTAdvisor to create multiple scan enables (if possible).

Examples

The following example turns on multiple scan enables during scan insertion for the three clock domains, clkA, clkB, and clkC. A maximum of six scan enable signals are created: clkA rising edge, clkA falling edge, clkB rising edge, clkB falling edge, clkC rising edge, and clkC falling edge.

```plaintext
add clocks 0 clkA clkB clkC
set system mode dft
set multiple scan_enables on
setup scan identification sequential atpg -percent 50
run
insert test logic -scan on -max_length 100
```

Using the preceding three clock domains example, you could merge the edges together for each clock (-Edge option on the Insert Test Logic command). The result would be two scan enable signals (and two scan chains) instead of six, one for the positive edge and one for the negative edge.

```plaintext
add clocks 0 clkA clkB clkC
set system mode dft
set multiple scan_enables on
setup scan identification sequential atpg -percent 50
run
insert test logic -scan on -max_length 100 -edge merge
```

Related Commands

- Insert Scan Chains
- Insert Test Logic
**Set Net Resolution**

Scope: Setup mode

**Usage**

```
SET NET Resolution Wire | And | Or
```

**Description**

Specifies the behavior of multi-driver nets.

The Set Net Resolution command specifies the behavior of non tri-state multi-driver nets. The default upon invocation of the tool is Wire, which requires all inputs be at the same value to achieve a value. If possible, you should specify the And or Or option, otherwise some loss of test coverage results.

**Arguments**

- **Wire**
  
  A literal that specifies for the tool to use unknown behavior for non tri-state multi-driver nets. This requires all inputs to be at the same value to achieve a value other than X. This is the default upon invocation of the tool.

- **And**
  
  A literal that specifies for the tool to use wired-AND behavior.

- **Or**
  
  A literal that specifies for the tool to use wired-OR behavior.

**Examples**

The following example specifies that the behavior of non tri-state multi-driver nets is wired-AND during the scan identification process.

```
set net resolution and
add clocks 0 clock
set system mode dft
run
report scan identification
```
Set Nonscan Handling

Scope: Setup modes

Usage

SET NOncan Handling {Check | Nocheck}

Description

Specifies whether to check the nonscan instances for scannability.

The Set Nonscan Handling command specifies whether the added nonscan instances are checked for scannability. Nonscan instances are defined by the Add Nonscan Instances command, Add Nonscan Models command, or the nonscan models due to no scan equivalents. These nonscan instances are not checked since the instances can not be selected for scan insertion.

You can enable checking on nonscan instances by using this command with the Check option.

Arguments

- Check | Nocheck
  A required literal that specifies whether to check the nonscan instances for scannability. The default upon invocation is Nocheck.

Examples

The following example shows how to check all added nonscan instances:

```
set nonscan handling Check
```

Related Commands

- Add Nonscan Instances
- Add Nonscan Models
- Set Drc Handling
Set Observe Threshold

Scope: All modes

Usage

SET OBserve Threshold integer

Description

Specifies the observability value for simulation-based test point identification.

The Set Observe Threshold command specifies minimum number of times a gate must be observed at a zero and a one state during simulation. The simulation is a set of pseudorandom patterns defined for test point insertion using the Setup Test_Point Identification command. Gates that do not meet the threshold are candidates for test points. You use this command primarily for Built-In Self-Test (BIST) circuitry.

Arguments

- integer

  A required integer, greater than or equal to 0, that specifies the minimum number of observations that you consider adequate during pseudorandom pattern test point identification. The default value upon invocation is 4.

Examples

The following example sets the threshold number to determine the observability effects during pseudorandom pattern test point identification:

set control threshold 2
set observe threshold 4
setup test_point identification -control 9 -obs 20 -patterns 32000
  -cshare 16 -oshare 16 -base simulation
set system mode dft
setup scan identification none
run

Related Commands

Set Control Threshold                     Setup Scan Identification
Set Random Patterns                       Setup Test_point Identification
Set Pattern Save

Scope: Setup and Bist modes

Usage

SET PAttern Save ON | OFF

Description

Enables the storing of BIST patterns.

The Set Pattern Save command specifies the storing of BIST patterns while fault simulation progresses. When pattern saving is turned off, DFTAdvisor uses less memory. However, in this case you cannot save patterns using the Save Patterns command.

You use this command primarily for BIST circuitry.

Arguments

• ON | OFF

A required literal that specifies whether to save BIST patterns during the fault simulation process. The default is on.

Examples

The following example turns off the saving of BIST patterns:

set pattern save off

Related Commands

Save Patterns
Set Random Patterns

Scope: All modes

Usage

SET RAndom Patterns integer

Description

Specifies the number of random patterns DFTAdvisor uses for simulation.
The Set Random Patterns command specifies how many random patterns you want DFTAdvisor to use for simulation.

Note

This command is only used for fault simulation in BIST mode and is not used for test point selection.

You use this command primarily for implementing Built-In Self-Test (BIST) circuitry.

Arguments

• integer

A required integer, greater than or equal to 0, that specifies the number of random patterns that you want DFTAdvisor to simulate. The default value upon invocation of DFTAdvisor is 1024.

Related Commands

Set Capture Clock
Set Control Threshold
Set Observe Threshold
**Set Scan Type**

Scope: Setup mode

**Usage**

SET Scan Type {Mux_scan | Lssd | Clocked_scan}

**Description**

Specifies the scan style design.

The Set Scan Type command specifies the scan style of a design which the scan insertion will create. If this command is not used, during the scan insertion process, the MUX scan style will be used.

**Arguments**

- **Mux_scan**
  
  A literal that specifies for DFTAdvisor to insert MUX type scan elements during the scan insertion process. This is the default behavior upon invocation.

- **Lssd**
  
  A literal that specifies for DFTAdvisor to insert LSSD type scan elements during the scan insertion process.

- **Clocked_scan**
  
  A literal that specifies for DFTAdvisor to insert Clocked-signal type scan elements during the scan insertion process.

**Examples**

The following example uses a MUX scan type design during scan insertion:

```
set scan type mux_scan
add clocks 0 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
insert test logic -max_length 10
```
Set Schematic Display

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Usage

SET SCHematic Display { -File filename | { -Compact | -NOCompact } | { -Query threshold | -NOQuery } | -Hide type | -Dspace { AUTO | number } } ...

DFTInsight Menu Path:
   Setup > Preferences

Description

Changes the default schematic display environment settings for DFTInsight.

The Set Schematic Display command only affects the environment and not the contents of the DFTInsight display. This command is optional when running DFTInsight because there are invocation defaults for all the settings. However, if you issue this command to change any of the defaults, you must specify at least one of the arguments.

If you have DFTInsight invoked, DFTInsight automatically updates the contents of the schematic viewing window when you change the settings. If you do not have DFTInsight invoked, DFTInsight uses the new settings when you do invoke the application.

Arguments

- **-File filename**

  A switch and string pair that changes the file location where the tool places, and where DFTInsight looks for, the display netlist. The default location upon invocation of DFTInsight is $MGC_HOME/tmp/dfti.<process#>/display.gn.

  If you leave the netlist at the default location, DFTInsight automatically deletes it when exiting the session. If you change the netlist location with the -File switch, the netlist remains persistent at the location you gave.
- **Compact**

A switch that specifies for DFTInsight to only display gates in the netlist that could have a logical impact on the output results. This is the default behavior upon invocation of DFTInsight.

If gate compaction is enabled, DFTI still maintains the inversion value of the signals. DFTI displays the inversion values by showing plus signs (+) and minus signs (-) next to the pin name, but only if there is an inversion difference between two displayed gates. The minus sign means that there is inversion between the two gates that are in the display.

- **-NOCompact**

A switch that specifies for DFTInsight to display all gates that are in the netlist, including buffers, inverters, Zval, and single-input bus gates. These types of gates do not affect the logical results and tend to clutter the display.

- **-Query threshold**

A switch and integer pair that specifies the maximum number of gates that you want DFTInsight to display without first asking if you want to continue. The threshold value is the number of total gates in the netlist, not just the number of compacted gates. The invocation default is -Query with a threshold value of 128 gates.

- **-NOQuery**

A switch that specifies for DFTInsight to display any size of display netlist you request. This can be a performance issue with very large gate counts.

- **-Hide type**

A switch and literal pair that specifies whether you want DFTInsight to place nets and port symbols on any input and output pins that are not driving (or being driven by) other instances that are contained within the display netlist.

The extra nets and ports tend to clutter the display, but they also allow you a way of selecting a single net to trace. The valid literals for the type parameter are as follows:

- **UO** — A literal that specifies to hide the unused output connections and to only display the unused input connections. This is the default upon invocation of DFTInsight.
UI — A literal that specifies to hide the unused input connects and to only display the unused output connections.

All — A literal that specifies to hide both the unused input and output connections.

None — A literal that specifies to show both the unused input and output connections.

- Dspace AUTO | number

A switch and value pair that specifies the maximum number of spaces that you want DFTInsight to use when displaying the pin data. When you use the -Dspace switch, you have the following two options:

AUTO — A literal that specifies for DFTInsight to automatically set the available space that the pin data requires for proper display. This is the default behavior upon invocation of DFTInsight.

number — A positive integer that specifies the maximum number of spaces that DFTInsight uses when displaying the pin data. If you specify a number that is smaller than the size of the pin data, DFTInsight could truncate the other data on the display.

Examples

The following example changes the default settings for the query threshold and the ports and nets that DFTInsight displays. The remaining options for compaction and the pin space size continue to use the default settings.

set schematic display -query 56 -hide none
open schematic display
add display instances 161 -backward

Related Commands

Add Display Instances          Report Gates
Analyze Drc Violation          Set Gate Report
Open Schematic Viewer
Set Screen Display

Scope: All modes

Usage

SET SCreen Display **ON** | **OFF**

Description

Specifies whether DFTAdvisor writes the transcript to the session window.

If you create a logfile with the Set Logfile Handling command, you may want to disable DFTAdvisor from writing the same information to the session transcript window.

Arguments

- **ON**
  
  A literal that specifies to enable the tool to write the session information to the transcript window. This is the default behavior upon invocation.

- **OFF**
  
  A literal that specifies to disable the tool from writing any of the session information to the transcript window, including error messages.

Examples

The following example shows how to use the logfile functionality to capture the transcript in a file and then disable DFTAdvisor from writing to the display:

```bash
set logfile handling /user/design/setup_file
set screen display off
```

Related Commands

- Report Environment
- Set Logfile Handling
**Set Sensitization Checking**

Scope: All modes

**Usage**

SET SEnsitization Checking **OFF | ON**

**Description**

Specifies whether DRC checking attempts to verify a suspected C3 rules violation.

The Set Sensitization Checking command specifies whether the DRC verifies that the path from the source and sink of a suspected C3 violation exists when the source and sink clocks are on and all other clocks are off. If sensitization checking is on and the paths associated with the violation meet these conditions, the DRC reports the C3 violation.

**Arguments**

- **OFF**
  
  A literal that disables the C3 DRC sensitization check. This is the default behavior upon invocation.

- **ON**
  
  A literal that enables the C3 DRC sensitization check.

**Related Commands**

- Set Drc Handling
Set Stability Check

Scope: Setup mode, before design rule checking

Usage

SET STability Check All_shift

Description

Specifies how the tool checks the effect of applying the shift procedure on non-scan cells.

When non-scan state elements control scan chain operation, DFTAdvisor needs to understand the values of these elements in order to perform scan chain tracing. For example, when a design contains BIST (that was implemented using LBISTArchitect), a shift counter -- composed of non-scan elements -- controls the scan chain shifting. This counter counts from zero to a predetermined number and then resets to zero. This predetermined number is related to the number of shifts applied by the load_unload procedure.

By default, DFTAdvisor simulates only one shift of the scan chain, which does not provide enough information about the non-scan elements and how they control scan shifting. Therefore, you should specify this command with the All_shift option to simulate all applications of the shift procedure or rules checking will not succeed.

Arguments

- All_shift

A literal that enables the tool to perform the most detailed level of checking. The shift procedure is simulated for as many applications as the load_unload procedure calls for. When you specify this option, it can significantly increase your run time. This option should be used with BIST.

Examples

The following example shows how to enable the detail checking which simulates the shift procedure more than once.

set stability check all_shift
Command Dictionary

Set System Mode

Scope: All modes

Usage

SET SYstem Mode Setup | Dft | Bist

Description

Specifies the next system mode for the tool to enter.

The Set System Mode command directs DFTAdvisor to a specific system mode, which includes scan insertion (Dft) or the default system mode of Setup.

When switching from the Setup mode to any other mode, DFTAdvisor builds a flat, gate-level simulation model. After the initial building of the flat, gate-level simulation model, if you return to Setup mode, issue any of the following commands, and then switch to another mode, a new simulation model is built:

- Add Nofaults
- Add Tied Signals
- Delete Nofaults
- Delete Tied Signals
- Set Internal Fault
- Set Internal Name
- Setup Tied Signals

Arguments

- Setup
  A literal that specifies for the tool to enter the Setup system mode.

- Dft
  A literal that specifies for the tool to enter the Scan Insertion system mode.

- Bist
  A literal that specifies for the tool to enter the LBISTArchitect system mode. This mode is only available if you have an LBISTArchitect license.
Examples

The following example will change the system mode so you can perform a scan identification run.

```plaintext
add tied signals 1 vcc
add tied signals 0 vss
add clocks 0 clock
set system mode dft
run
report scan identification
```
Set Test Logic

Scope: Setup mode

Usage

SET TEST Logic {-Set {ON | OFF} | -REset {ON | OFF} | -Clock {ON | OFF} | -Tristate {ON | OFF} | -RAm {ON | OFF}}...

Description

Specifies which types of control lines DFTAdvisor makes controllable during the DFT rules checking.

The Set Test Logic command specifies whether DFTAdvisor makes set, reset, clock, enable, or write control lines controllable. This, in turn, makes them scannable by inserting test logic to those lines when you insert scan chains. Before inserting scan chains, you must specify the cell models with the Add Cell Models command. You can use the Report Dft Check command to display the lines where you can insert test logic to make the lines controllable.

The default behavior upon invocation of DFTAdvisor is for all the Set Test Logic command switches to be Off. You can use the Report Environment command to display the current test logic settings.

If all enable lines of a bus are driven purely by combinational logic, then no additional test logic is added since there will not be a problem of contention due to scan shifting.

Arguments

• -Set ON | OFF
  A switch and literal pair that specifies whether DFTAdvisor makes the set lines controllable during the DFT rules checking. The default upon invocation of DFTAdvisor is Off.

• -REset ON | OFF
  A switch and literal pair that specifies whether DFTAdvisor makes the reset lines controllable during the DFT rules checking. The default upon invocation of DFTAdvisor is Off.
• **-Clock ON | OFF**

A switch and literal pair that specifies whether DFTAdvisor makes the clock lines controllable during the DFT rules checking. The default upon invocation of DFTAdvisor is Off.

• **-Tristate ON | OFF**

A switch and literal pair that specifies whether DFTAdvisor makes the tri-state enable lines controllable during the DFT rules checking. This test logic is only active during scan chain shifting. The default upon invocation of DFTAdvisor is Off.

If you set the -Tristate option On, DFTAdvisor uses the scan enable to turn on one of the tri-state gates driving the bus, and turns off the rest of the tri-state gates. If tri-state gates driving the same bus already have one-hot logic controlling the enable lines of these gates, no additional test logic is added.

• **-RAm ON | OFF**

A switch and literal pair that specifies whether DFTAdvisor makes the write control lines controllable during the DFT rules checking. This option does not handle the set and reset lines of the RAM. The default upon invocation of DFTAdvisor is Off.

**Examples**

The following example checks the set and clock lines of uncontrollable memory elements and makes them controllable with the addition of test logic:

```
add clocks 0 clk
set test logic -set on -clock on
set system mode dft
report dft check
add cell models and2 -type and
add cell models or2 -type or
add cell models mux21h -type mux s a b
add cell models nor2 -type nor
report cell models
insert test logic
```
Related Commands

- Add Cell Models
- Delete Cell Models
- Report Cell Models
- Set Latch Handling
Set Trace Report

Scope: All modes

Usage

SET TRace Report **OFF** | **ON**

Description

Specifies whether the tool displays gates in the scan chain trace.

The Set Trace Report command controls whether the tool displays all of the gates in the scan chain trace during rules checking.

Arguments

- **OFF**
  
  A literal that specifies for the tool to not display gates in the scan chain trace. This is the default behavior upon invocation.

- **ON**
  
  A literal that specifies for the tool to display gates in the scan chain trace during rules checking.

Examples

The following example displays the gates in the scan chain trace during rules checking:

```
add clocks 0 clock
add scan groups group1 scanfile
add scan chains chain1 group1 indata2 outdata4
set trace report on
set system mode dft
```

Related Commands

- Add Scan Chains
- Report Scan Chains
**Set Zoom Factor**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

**Usage**

SET ZOOM Factor *scale_factor*

DFTInsight Menu Path:
- Zoom Scale menu in the Icon bar.

**Description**

Specifies the scale factor that the zoom icons use in the DFTInsight Schematic View window.

The Set Zoom Factor command only affects the behavior of the Zoom In and Zoom Out icons in the DFTInsight Schematic View window. The zoom factor does not affect the behavior of the Zoom In or Zoom Out commands.

![Figure 2-5. Zoom In and Zoom Out Icons](image)

**Arguments**

- *scale_factor*
  
  A required integer that specifies the multiplication factor used to determine how much to enlarge or reduce the selected objects in the Schematic View window.
Setup LFSRs

Scope: All modes
Prerequisites: This command intended for an LBISTArchitect design flow.

Usage

SETup LFsrs { -Both | -Serial | -Parallel } { -Out | -In }

Description

Changes the shift_type and tap_type default setting for the Add LFSRs and Add LFSR Taps commands.

The Setup LFSRs command controls the default setting for the shift_type and tap_type switches. You specify the LFSR’s shift technique by using one of the following shift_type switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following tap_type switches: -Out or -In. When you change one or both of the default settings, all future Add LFSRs and Add LFSR Taps commands use the new default.

Arguments

The following lists the three shift_type switches of which you can choose only one:

- **-Both** — A switch specifying that the LFSR shifts both serially and in parallel. This is the default behavior upon invocation.
- **-Serial** — A switch specifying that a serial shift LFSR shifts a number of times equal to the length of the longest scan chain for each scan pattern.
- **-Parallel** — A switch specifying that a parallel shift LFSR shifts once for each scan pattern.

The following lists the two tap_type switches of which you can only choose one:

- **-Out** — A switch that places the exclusive-or taps outside the register path. This is the default upon invocation.
- **-In** — A switch that places the exclusive-or taps in the register path.
Examples

The following example changes the default \textit{shift\_type} setting to Serial and the default \textit{tap\_type} switch to In:

\begin{verbatim}
setup lfsrs -serial -in
add lfsrs lfsr1 prpg 5 13
add lfsrs lfsr2 prpg 5 11
add lfsr taps lfsr1 2 3
add lfsr taps lfsr2 3 4
set system mode dft
\end{verbatim}

Related Commands

Add LFSRs
Add LFSR Taps
Delete LFSR Taps
Report LFSRs
Delete LFSRs
Setup Output Masks

Scope: Setup mode

Usage

SETup OUtput Masks **OFF** | { **ON** [-BidiExclude] [-LbistExclude] [-Exclude primary_output_pin...]

Description

Sets the default mask for all output and bi-directional pins.

The Setup Output Masks command masks all the output and bi-directional pins specified. This default mask is present on the output pin, unless overridden by the Add Output Masks command.

DFTAdvisor uses primary output pins as the observe points during the scan and test point identification process. When you mask a primary output pin, you inform DFTAdvisor to mark that pin as an invalid observation point during the scan and test point identification process.

You can exclude bi-directional pins, LBISTArchitect scan output pins, and any specified pins.

You use the Off literal to remove all default masks defined with this command.

You can add a hold value to a default mask with the Add Output Masks command, or remove a hold value using the Delete Output Masks command. You can also use the Add Output Masks command to add a mask to any of the output pins excluded with the Setup Output Masks command.

Arguments

- **OFF | ON**
  A required string that specifies to set or remove the current default mask setting for all primary output pins. When turning masking on, you can exclude pins from the mask. The tool invocation default setting is no masks.

- **-BidiExclude**
  An optional switch that specifies to exclude bi-directional pins from the mask setting.
• -Lbist_exclude
  An optional switch that specifies to exclude LBISTArchitect scan output pins from the mask setting.

• -Exclude primary_output_pin...
  An optional switch and repeatable string that specifies to exclude the specified primary output pins from the mask setting.

Examples

The following example defines the default mask for all but the LBISTArchitect scan output pins, then adds two additional pin masks with a hold value of 1:

```
setup output masks on -lbist_exclude
add output masks out1 out2 -hold 1
```

Related Commands

Add Output Masks                      Report Output Masks
Delete Output Masks
Setup Pin Constraints

Scope: Setup mode

Usage

SETup PIn Constraints None | {{C0 | C1 | CZ | CX} [-Bidi_exclude]
[-Lbist_exclude] [-Exclude primary_input_pin...]]

Description

Sets the default pin constraint value for all input and bi-directional pins.

The Setup Pin Constraints command constrains all input and bi-directional pins to
the specified default value. You use this command to constrain the input during
BIST mode in order to avoid the inputs being tied to X. This default value is
present on the input pin, unless overridden by the Add Pin Constraints command.

You can exclude bi-directional pins, LBISTArchitect related control and data
pins, and any specified pin.

You use the None literal to remove all default settings defined with this command.

Arguments

- None
  A required string that specifies to remove all the current default constraints for
  all primary input and bi-directional pins. You must specify this literal or one of
  the “C” literals. This is the tool invocation default.

- C0 | C1 | CZ | CX
  A literal specifying the default constant value constraint for the primary_input
  pins, except for any pins excluded by the specified exclude options. You must
  specify one of these literals or the None literal. The constraint choices are as
  follows:
    - C0 — A literal that specifies application of the constant 0 to the chosen
          primary input pins.
    - C1 — A literal that specifies application of the constant 1 to the chosen
          primary input pins.
**CZ** — A literal that specifies application of the constant Z (high-impedance) to the chosen primary input pins.

**CX** — A literal that specifies application of the constant X (unknown) to the chosen primary input pins.

- **-Bidi_exclude**
  An optional switch that specifies to exclude bi-directional pins from the setup setting.

- **-Lbist_exclude**
  An optional switch that specifies to exclude LBISTArchitect related input pins from the setup setting. These pins include clocks, RAM/ROM read signals, RAM/ROM write signals, sets, resets, and known scan inputs.

- **-Exclude primary_input_pin...**
  An optional switch and repeatable string that specifies to exclude the specified primary input pins from the setup setting.

**Examples**

The following example defines the default pin constraints for all but the LBISTArchitect related control and data pins, then adds two additional pin constraints that override the default:

```
setup pin constraints c0 -lbist_exclude
add pin constraints kgmt c1
add pin constraints ckgmt c1
```

**Related Commands**

- Add Pin Constraints
- Add Seq_transparent Constraints
- Analyze Input Control
- Delete Pin Constraints
- Report Pin Constraints
Setup Scan Identification

Scope: All modes

Usage

SETup SCan Identification

**Full_scan**

{**Clock_sequential** [-Depth *integer*]} | 
{**SEQ_transparent** [-Reconvergence {ON | OFF}]} | 
{**Partition_scan** [-Input_threshold {integer | Nolimit}]
[-Output_threshold {integer | Nolimit}] | 
{**SEQUential**
{**Atpg** [{-Percent *integer*} | {-Number *integer*}]
[-Internal | -External filename]
[-COntrollability *integer*] [-Observability *integer*]
[-Backtrack *integer*] [-CYcle *integer*] [-Time *integer*]
[-Min_detection floating_point]} | 
{**AUtomatic** [{-Percent *integer*} | {-Number *integer*}]
{**SCoap** [-Percent *integer* | -Number *integer*]} | 
{**STructure** [{-Percent *integer*} | {-Number *integer*}]
[-Loop {ON | OFF}]
[-Self_loop {integer | Nolimit}]
[-Depth {integer | Nolimit}]} | 

**None**

Description

Specifies the scan identification methodology and amount of scan that DFTAdvisor is to consider during the identification run.

The Setup Scan Identification command controls the scan identification run. You can choose from many scan identification methods. After you choose a method, you can specify the options that apply to that method.

Full Scan Arguments

- **Full_scan**

A required literal that specifies to use full scan for scan identification. Full scan is the fastest identification method, converting all scannable sequential elements to scan. You can use FastScan for ATPG on full scan designs. This is the default upon invocation of the tool. For more information on full scan, refer to "Understanding Full Scan" in the Scan and ATPG Process Guide.
Clock Sequential Arguments

- **Clock_sequential**
  A required literal that specifies to use clock sequential techniques for scan identification. This method selects scannable cells by cutting sequential loops and limiting sequential depth based on the -Depth switch. Typically, this method is used to create structured partial scan designs that can use FastScan’s clock sequential ATPG algorithm. For more information on clock sequential scan, refer to “FastScan Handling of Non-Scan Cells” in the Scan and ATPG Process Guide.

- **-Depth integer**
  An optional switch and integer pair that specifies the maximum sequential depth on any sequential path. The maximum depth is 255. The default depth is 16.

Sequential Transparent Arguments

- **SEQ_transparent**
  A required literal that specifies to use the sequential transparency identification method for scan. Note that this technique is useful for data path circuits. Scan cells are selected such that all sequential loops, including self loops, are cut. For more information on sequential transparent scan, refer to “FastScan Handling of Non-Scan Cells” in the Scan and ATPG Process Guide.

- **-Reconvergence {ON | OFF}**
  An optional switch and literal pair that specifies to remove sequential reconvergent paths by selecting a scannable instance on the sequential path for scan. The default is ON.

Partition Scan Arguments

- **Partition_scan**
  A required literal that specifies to use partition scan for controllability and observability of embedded blocks. You can also set threshold limits to control the overhead sometimes associated with partition scan identification. For example, overhead extremes may occur when DFTAdvisor identifies a large number of partition cells for a given uncontrollable primary input or unobservable primary output. By setting the partition threshold limit for
primary inputs (-Input_threshold switch) and primary outputs
(-Output_threshold switch), you maintain control over the trade-off of whether
to scan these partitioned cells or, instead, insert a controllability/observability
scan cell.

When DFTAdvisor reaches the specified threshold for a given primary input or
primary output, it terminates the partition scan identification process on that
primary input or primary output and unmarks any partition cell identified for
that pin. For more information on partition scan, refer to “Understanding
Partition Scan” in the Scan and ATPG Process Guide.

• -Input_threshold {integer | Nolimit}
  
  An optional switch and integer or literal pair that specifies the maximum
  number of partition scan cells for any given uncontrollable input. The default
  is nolimit.

• -Output_threshold {integer | Nolimit}
  
  An optional switch and integer or literal pair that specifies the maximum
  number of partition scan cells for any given unobservable output. The default
  is nolimit.

Sequential Using ATPG Arguments

• SEQUential
  
  A required literal that specifies to use partial scan for scan identification.
  Partial scan (sequential) results in a subset of scannable sequential elements
  being converted to scan. Partial scan requires that you choose which algorithm
to use (ATPG, Automatic, SCOAP, or Structure) and how many scan elements
to identify. You can use FlexTest for ATPG on partial scan designs. For more
information on partial scan, refer to “Understanding Partial Scan” in the Scan
and ATPG Process Guide.

• Atpg
  
  An optional literal that specifies for DFTAdvisor to use the ATPG-based
technique for partial scan selection. This method selects scan cells using the
sequential ATPG algorithm of FlexTest. This is the default partial scan
method.
• {-Percent integer} | {-Number integer}

An optional switch and integer pair that specifies the maximum percentage of scan (based on the total number of sequential elements in the design) or absolute number of sequential instances that you want to identify as scan. By default, the number of scan cells the tool can select is 100 percent (which means that there is no limit). When specifying the percentage, you are providing DFTAdvisor an absolute maximum percentage of scan cells it can choose. Often, it will choose less than you specify if it can do so and still meet the other criteria of selection.

• -Internal

An optional switch that specifies for DFTAdvisor to base scan identification on the internally generated fault list. This is the default.

• -External filename

An optional switch and string pair that specifies the fault file on which you want DFTAdvisor to base scan identification. The file must contain the user-defined fault list for identifying critical flip-flops that you want to convert to scan flip-flops.

• -COntrollability integer

An optional switch and integer pair that specifies the desired percentage of controllability test coverage. DFTAdvisor continues the scan identification process until it either reaches the desired test coverage or no productive scan candidate are available. The default upon invocation of DFTAdvisor is 100 percent.

• -Observability integer

An optional switch and integer pair that specifies the desired percentage of observability test coverage. DFTAdvisor continues the scan identification process until it either reaches the desired test coverage or no productive scan candidate are available. The default upon invocation of DFTAdvisor is 100 percent.

• -Backtrack integer

An optional switch and integer pair that specifies the number of conflicts DFTAdvisor encounters before aborting the target fault. The default upon invocation of DFTAdvisor is 30 conflicts.
Setup Scan Identification

• -CYcle integer
  An optional switch and integer pair that specifies the number of test cycles DFTAdvisor encounters before aborting the target fault. The default upon invocation of DFTAdvisor is 16 test cycles.

• -Time integer
  An optional switch and integer pair that specifies the CPU time in seconds that DFTAdvisor uses before aborting the target fault. The default upon invocation of DFTAdvisor is 100 seconds of CPU time.

• -Min_detection floating_point
  An optional switch and floating point pair that specifies the minimum percentage of test coverage that a scan cell must provide. If a scan cell does not detect at least the specified minimum percentage of faults, DFTAdvisor does not select the cell for scan. The default upon invocation of DFTAdvisor is 0.01 percent.

Sequential Using Automatic Arguments

• SEQUential
  A required literal that specifies to use partial scan for scan identification. Partial scan (sequential) results in a subset of scannable sequential elements being converted to scan. Partial scan requires that you choose which algorithm to use (ATPG, Automatic, SCOAP, or Structure) and how many scan elements to identify. You can use FlexTest for ATPG on partial scan designs. For more information on partial scan, refer to “Understanding Partial Scan” in the Scan and ATPG Process Guide.

• AUtomatic
  An optional literal that specifies for DFTAdvisor to use the automatic technique for partial scan selection. This method selects scan cells using a combination of several scan selection techniques. The goal is to select the minimum set of best scan candidates needed to achieve high fault coverage.

• {-Percent integer} | {-Number integer}
  An optional switch and integer pair that specifies the maximum percentage of scan (based on the total number of sequential elements in the design) or absolute number of sequential instances that you want to identify as scan. By
default, automatic scan selection analyzes the circuit and attempts to identify the minimum amount of scan needed to achieve high fault coverage. However, if a limit is set using either of those two switches, DFTAdvisor attempts to select the best scan cells within the limit.

It is recommended that during the first scan selection and ATPG iteration, you use the default to allow the tool to determine the amount of scan needed. Then based on the ATPG results and how they compare to the required test coverage criteria, you can specify the exact amount of scan to select. The amount of scan selected in the first (default) iteration can be used as a reference point for determining how much more or less scan to select in subsequent iterations (i.e. what limit to specify).

**Sequential Using SCOAP Arguments**

- **SEQUential**
  
  A required literal that specifies to use partial scan for scan identification. Partial scan (sequential) results in a subset of scannable sequential elements being converted to scan. Partial scan requires that you choose which algorithm to use (ATPG, Automatic, SCOAP, or Structure) and how many scan elements to identify. You can use FlexTest for ATPG on partial scan designs. For more information on partial scan, refer to “Understanding Partial Scan” in the *Scan and ATPG Process Guide*.

- **SCoap**
  
  An optional literal that specifies for DFTAdvisor to use the SCOAP-based technique for partial scan selection. SCOAP-based selection is typically faster than ATPG-based selection, and produces an optimal set of scan candidates.

- **{-Percent integer} | {-Number integer}**
  
  An optional switch and integer pair that specifies the percentage of scan (based on the total number of sequential elements in the design) or absolute number of sequential instances that you want to treat as scan instances. By default, the number of scan cells the tool selects is 50 percent.
Sequential Using Structure Arguments

- **SEQUential**
  A required literal that specifies to use partial scan for scan identification. Partial scan (sequential) results in a subset of scannable sequential elements being converted to scan. Partial scan requires that you choose which algorithm to use (ATPG, Automatic, SCOAP, or Structure) and how many scan elements to identify. You can use FlexTest for ATPG on partial scan designs. For more information on partial scan, refer to “Understanding Partial Scan” in the Scan and ATPG Process Guide.

- **STructure**
  An optional literal that specifies for DFTAdvisor to use structure-based scan selection techniques. These techniques include loop breaking, self-loop breaking, and limiting the design’s sequential depth.

- **{-Percent integer} | {-Number integer}**
  An optional switch and integer pair that specifies the maximum percentage of scan (based on the total number of sequential elements in the design) or absolute number of sequential instances that you want to identify as scan. By default, the number of scan cells the tool can select is 100 percent (which means that there is no limit). When specifying the percentage, you are providing DFTAdvisor an absolute maximum percentage of scan cells it can choose. Often, it will choose less than you specify if it can do so and still meet the other criteria of selection.

- **-Loop {ON | OFF}**
  An optional switch and literal pair that specifies to cut global loops by inserting scan instances. The default is ON.

- **-Self_loop {integer | Nolimit}**
  An optional switch and integer or literal pair that specifies the maximum number of consecutive self-loops allowed to remain on any sequential path. The default is 8.

- **-Depth {integer | Nolimit}**
  An optional switch and integer or literal pair that specifies the maximum sequential depth allowed to remain on any sequential path. The default is 16.
No Scan Identification Argument

- None

A required literal that specifies to not perform scan identification. You use this option in combination with the Add Test Points or Setup Test_point Identification command if you only want to insert test points (and not scan) in your design. If you want to insert both test points and scan, you should always do the scan identification before the test point identification to ensure an optimal test point selection. For more information on test points, refer to “Understanding Test Points” in the Scan and ATPG Process Guide.

Examples

The following example sets up scan identification to identify a maximum of 50 percent of the design’s total number of sequential instances for scan:

```plaintext
set system mode dft
setup scan identification sequential atpg -percent 50
run
```

The following example sets up for full scan:

```plaintext
setup scan identification full_scan
run
```

The following example sets up for a test point only run:

```plaintext
setup scan identification none
setup test_point identification -control 10 -observe 5
run
```

Related Commands

- Add Test Points
- Report Scan Identification
- Run
- Setup Test_point Identification
- Write Scan Identification
Setup Scan Insertion

Scope: All modes

Usage

SETup SCan INsertion [{-SEN name | -TEn name} {-Active {High | Low} }] [-TClk name] [-SClk name] [-SMclk name] [-SSclk name] [{ {-SET name} | {-RESet name} | {-Write name} | {-REAd name} }... {-Muxed | -Disabled | -Gated }]

Description

Sets up the parameters for the Insert Scan Chains and Insert Test Logic commands.

The Setup Scan Insertion command allows you to either change or set the pin names that DFTAdvisor assigns during the Insert Scan Chains and Insert Test Logic commands. The default names upon invocation of DFTAdvisor which you can change are listed in Table 2-4.

Table 2-4. Scan Insertion Invocation Default Pin Names

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
<th>Default Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-SEN</td>
<td>Scan Enable for mux-DFF type</td>
<td>scan_en</td>
</tr>
<tr>
<td>-TEn</td>
<td>Test Enable for all scan types</td>
<td>test_en</td>
</tr>
<tr>
<td>-TClk</td>
<td>New Scan Clock for all scan types</td>
<td>test_clk</td>
</tr>
<tr>
<td>-SClk</td>
<td>Scan Clock for clock-scan type</td>
<td>scan_clk</td>
</tr>
<tr>
<td>-SMclk</td>
<td>Scan Master Clock for LSSD type</td>
<td>scan_mclck</td>
</tr>
<tr>
<td>-SSclk</td>
<td>Scan Slave Clock for LSSD type</td>
<td>scan_sclck</td>
</tr>
<tr>
<td>-SET</td>
<td>Global Set</td>
<td>scan_set</td>
</tr>
<tr>
<td>-RESet</td>
<td>Global Reset</td>
<td>scan_reset</td>
</tr>
<tr>
<td>-Write</td>
<td>RAM Write Control</td>
<td>write_clk</td>
</tr>
<tr>
<td>-REAd</td>
<td>RAM Read Control</td>
<td>read_clk</td>
</tr>
</tbody>
</table>
DFTAdvisor uses the set and reset names when gating the set and reset pins of flip-flops or latches. Furthermore, you can use the -Disable or -Muxed switches to specify whether DFTAdvisor uses an AND gate or MUX gate when performing the gating. If you specify the -Disabled option, then for gating purposes DFTAdvisor uses the test enable signal to disable the set and reset inputs of flip-flops. If you specify the -Muxed option, then for muxing purposes DFTAdvisor uses any set and reset pins which are defined as clocks to multiplex with the original signal.

DFTAdvisor uses the write control and read control pins when inserting test logic for RAMs. After you have added the write control lines in the Setup system mode and switched to the DFT system mode, the Design Rules Checker checks whether the RAMs can be held off when the write control pin is off. Those RAMs which cannot be held off are candidates for inserting test logic.

If the new scan design is intended for FastScan, and if the write clock input of the RAM is not controllable, it will be muxed out by a new write clock and the selector to the mux will be the test enable pin.

If the new scan design is intended for FlexTest, and if the original write clock is not to be muxed and the RAM is just to hold during the shifting of the scan chain data, you need to specify the -Disabled option. Thus, DFTAdvisor will use the scan enable pin to gate the original write clock input of the RAM.

In addition, you can use instance pin pathnames to define the scan enable, test enable, etc. pin names. If you specify an internal pin pathname and DFTAdvisor cannot trace through a simple path (only through inverters or buffers) back to a primary input/output of the design, DFTAdvisor cannot generate the test procedure file and the dofile with the Write Atpg Setup command. Therefore, you must manually develop these two files before running ATPG with either FastScan or FlexTest.

**Arguments**

- **-SEN name**

  An optional switch and string pair specifying the leading scan enable pin name that you want the scan insertion process to use when creating the scan enable. The default scan enable pin name upon invocation of DFTAdvisor is “scan_en”. If you specify for DFTAdvisor to create multiple scan enables
(using the `Set Multiple Scan_enables` command), then this name is used as the prefix for each scan enable name (`nameN`).

You can use the `-Active` switch to specify whether the scan enable pin is active low or active high

- `-TEn name`

An optional switch and string pair specifying the leading test enable pin name that you want the scan insertion process to use when creating the test enable. The default test enable pin name upon invocation of DFTAdvisor is “test_en”.

You can use the `-Active` switch to specify whether the test enable pin is active low or active high

- `-Active High | Low`

An optional switch and literal pair that specifies whether the scan enable or test enable pin is active high or active low. You can only use this switch with the `-Sen` or `-Ten` switch. The default upon invocation of DFTAdvisor is active high.

- `-TClk name`

An optional switch and string pair specifying the name of a new scan clock pin that you want DFTAdvisor to create if a scan clock is not already predefined.

If you have predefined a scan clock, DFTAdvisor will use it. Otherwise, DFTAdvisor creates a new pin with the specified name during the scan insertion process. If neither of these conditions exist, DFTAdvisor creates a new test clock pin using the invocation default name of “test_clk”.

- `-SClk name`

An optional switch and string pair specifying the leading scan clock pin name that you want the scan insertion process to use when creating the scan clock. DFTAdvisor only creates the scan clock pin for the scan clock port in the dual-port type mux scan. The default scan clock pin name upon invocation of DFTAdvisor is “scan_clk”.

- `-SMclk name`

An optional switch and string pair specifying the leading scan master clock pin name that you want the scan insertion process to use when creating the scan
master clock. The default scan master clock pin name upon invocation of DFTAdvisor is “scan_mclk”.

• **-SSclk name**
  An optional switch and string pair specifying the leading scan slave clock pin name that you want the scan insertion process to use when creating the scan slave clock. The default scan slave clock pin name upon invocation of DFTAdvisor is “scan_sclk”.

• **-SET name**
  An optional switch and string pair specifying the set pin name that you want DFTAdvisor to use for the flip-flop or latch when inserting test logic. The default set pin name upon invocation of DFTAdvisor is “scan_set”.

• **-RESet name**
  An optional switch and string pair specifying the reset pin name that you want DFTAdvisor to use for the flip-flop or latch when inserting test logic. The default reset pin name upon invocation of DFTAdvisor is “scan_reset”.

• **-Write name**
  An optional switch and string pair specifying the write control pin name that you want DFTAdvisor to use for the RAM when inserting test logic. The default write control pin name upon invocation of DFTAdvisor is “write_clk”.

• **-REAd name**
  An optional switch and string pair specifying the read control pin name that you want DFTAdvisor to use for the RAM when inserting test logic. The default read control pin name upon invocation of DFTAdvisor is “read_clk”.

• **-Muxed**
  An optional switch that specifies for DFTAdvisor to mux the set, reset, write control, or read control lines. This is the default. If you specify the -Muxed option, then for muxing purposes DFTAdvisor will multiplex any set and reset pins which are defined as clocks with the original signal.

• **-Disabled**
  An optional switch that specifies for DFTAdvisor to gate the set, reset, write clock, or read clock lines. If you specify the -Disabled option, then for gating
purposes DFTAdvisor will use the test enable signal to disable set and reset inputs of flip-flops and scan enable signal to disable the write and read clocks.

- -Gated

An optional switch that specifies for DFTAdvisor to gate the set, reset, write control, or read control lines. If you specify the -Gated option, then for gating purposes DFTAdvisor will use any set and reset pins defined as clock or use the write and read clocks to disable the set and reset inputs of flip-flops.

**Examples**

The following example renames the test enable pin name to test_en_L and specifies for the pin to be active low during the scan insertion process:

```
add clocks 0 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
setup scan insertion -ten test_en_L -active low
insert test logic -max_length 10
```

The following example shows how to set different controls using successive Set Scan Insertion commands:

```
setup scan insertion -ten TEST_MODE -active high // test enable
setup scan insertion -sen SCAN_MODE // scan enable
setup scan insertion -tclk TEST_CLK // test clock
setup scan insertion -reset RESET -mux // global reset
setup scan insertion -set ATPG_SET -disable // global set
setup scan insertion -write ATPG_WRT_INH -mux // RAM write controls
setup scan insertion -read ATPG_READ_INH -mux // RAM read controls
```

**Related Commands**

Insert Test Logic
Setup Scan Pins

Scope: All modes

Usage

SETup SCan Pins {Input | Output} [-INDexed | -Bused] [-Prefix base_name] [-INIitial index#] [-Modifier incr_index#] [-Suffix suffix_name]

Description

Changes the scan-in or scan-out pin naming parameters to index or bus format.

The Setup Scan Pins command specifies the index or bus naming conventions for scan-in or scan-out pins. The Report Environment command displays the names and values that DFTAdvisor uses for scan-in and scan-out pins when inserting scan chains. Once you have inserted scan chains, you can use the Report Scan Chains command to display the chosen index or bus naming convention for the scan-in pins and/or scan-out pins.

Indexed names are in the form:

    base_name + index# + suffix_name

Bused names are in the form:

    base_name + [index#]

Arguments

- **Input**
  
  A literal specifying that DFTAdvisor apply the index or bus format on the scan-in pins.

- **Output**
  
  A literal specifying that DFTAdvisor apply the index or bus format on the scan-out pins.

- **-INDexed**
  
  An optional switch specifying that DFTAdvisor apply the index format to the scan-in or scan-out pin names. This is the default.
• **-Bused**
  An optional switch specifying that DFTAdvisor apply the bus format to the scan-in or scan-out pin names.

• **-Prefix base_name**
  An optional switch and string pair that specifies the root name of the scan-in or scan-out pin. The default name is `scan_in`.

• **-INITial index#**
  An optional switch and integer pair that specifies the initial index value of the scan-in or scan-out pin name. The default value is 1.

• **-Modifier incr_index#**
  An optional switch and integer pair specifying the incremental value that to add to the `index#` when creating additional names with the same `base_name`. The default is 1.

• **-Suffix suffix_name**
  An optional switch and string pair specifying the name that you want to place after the `index#`. DFTAdvisor only uses this for indexed naming. The default is `null`.

### Examples

The following example configures scan insertion to use bus names for the scan-in pins and scan-out pins, with the index number starting at 5 and incrementing by 2 for scan-in pins, and the index number starting at 4 and incrementing by 2 for scan-out pins:

```
add clocks 0 clock
set system mode dft
run
setup scan pins input -bused -prefix scin -initial 5 -modifier 2
setup scan pins output -bused -prefix scout -initial 4 -modifier 2
insert test logic -number 7
```

### Related Commands

- Insert Test Logic
Setup Test_point Identification

Scope: Setup (all options) and Dft (Scoap and Simulation options only) modes

Prerequisites: If you want only test points (and not scan) identified, use the Setup Scan Identification command’s None option. Also, the Simulation and Multiphase base options are only available if you have a LBISTArchitect license.

Usage

SETup TEst_point IDentification [-COntrol integer] [-OBserve integer] [-Verbose | -NOVerbose] [-BAse {
    {SCoap [-Internal | {-External filename}] | 
    {SImulation [-CShare integer] [-OShare integer] [-PATterns integer] 
        [-PASs_size integer]} | 
    {Multiphase [-Test_coverage percent] [-PHases integer] 
        [-BPc_threshold integer] [-Sig_prob_threshold percent] 
        [-NUm_detections integer] [-OP_cost multiplier] [-PATterns integer] 
        [-Rcp_cost multiplier]}]]

Description

Specifies the number of control and observe test points that DFTAdvisor flags during the identification run.

Test points allow you to gain access into the design to aid in detecting the undetectable faults, which increases the design’s fault coverage. Test points may translate into additional input or output pins or scan cells at the chip level.

Control test points allow you to gain access to the inputs of a gate (or multiple gates) where faults are undetectable. Usually the reason these faults are undetectable is because the ATPG tool cannot simultaneously force the correct combination of values onto the inputs. By creating a test point at that location, you give the ATPG tool direct control over the input values of that gate, which then gives the tool the ability to force (control) the necessary combination of input values to detect the previously undetectable faults.

Observe test points allow you to gain access to the outputs of a gate (or multiple gates) where faults are undetectable. With observability problems, usually the reason these faults are undetectable is because the effects of this fault never propagate to a primary output or other observation point. A fault cannot be
detected unless one or more primary outputs shows the fault effect. By creating a test point at that location, you allow the direct access to the fault effect, which changes the fault from an unobservable (undetected) fault to a detected fault.

Once you have enabled DFTAdvisor to identify the optimum test point locations (with the Setup Scan Identification command) and set the number of each type of test point that DFTAdvisor is to locate (with the Setup Test_point Identification command), you need to enter Dft mode (with the Set System Mode Dft command). After successfully entering Dft mode, you perform the test point analysis with the Run command. While performing the analysis, DFTAdvisor lists the test point locations that DFTAdvisor identifies.

For more information on controllability and observability, refer to the *Scan and ATPG Process Guide*. For information on multiphase test point insertion, refer to “Multiphase Test Point Insertion Analysis” in the *Built-in Self-Test Process Guide*.

**Arguments**

- **-COntr01 integer**
  An optional switch and integer pair that specifies how many test points you want DFTAdvisor to identify to aid in increasing the controllability of the design. The default upon invocation of DFTAdvisor for identifying test points for controllability is 0.

- **-OBServe integer**
  An optional switch and integer pair that specifies how many test points you want DFTAdvisor to identify to aid in increasing the observability of the design. The default upon invocation of DFTAdvisor for identifying test points for observability is 0.

- **-V erb0se | -NOV erb0se**
  An optional switch that specifies the amount of information that DFTAdvisor displays during test point generation.

- **-B ase SCoap | SImulation | Multiphase**
  An optional switch and literal pair that determines the pin selection technique that you want DFTAdvisor to use when calculating their controllability and observability values. It is based on this selection that DFTAdvisor identifies
the test point candidates and inserts the test points. During a DFTAdvisor session, once you select a technique, you should not change. Doing so may result in unpredictable results. The following lists the techniques available:

**SCoap** — A literal specifying that DFTAdvisor use the Scoap-based technique to calculate the controllability and observability values for all the pins in the flattened design. You use this literal primarily for non-BIST applications.

**SIMulation** — A literal specifying that DFTAdvisor use the simulation-based technique to calculate the controllability and observability values for each pin in the flattened design. This technique uses one set of random patterns to calculate the controllability and observability. You use this literal primarily for BIST applications and only if you have an LBISTArchitect license.

**Multiphase** — A literal specifying that DFTAdvisor use the Multi-phase-based test point insertion technique to calculate the controllability and observability values for each pin in the flattened design. This technique divides the total number of random patterns into the same number of groups as you specify phases (using the -Phases argument). You can only specify this option if you are in Setup mode and only if you have an LBISTArchitect license.

• **-Internal | -External filename**
  
  An optional switch that specifies which faults to use when performing SCOAP-based test point selection.

  - **-Internal** — A switch that specifies to consider all faults. This is the default.
  - **-External filename** — A switch and string that specifies to use only those faults listed in the specified faults list file for evaluating the benefit of a test point. Detected, redundant, and unused fault found in the file are ignored. Any other faults previously loaded are discarded. Typically, this fault list file is saved after running ATPG. This file only needs to be set once, and it is used by all SCOAP test point selection methods until it is changed (with -Internal or -External).

• **-CShare integer**
  
  An optional switch and integer pair that you can only use with the Simulation literal to specify the maximum number of control points you want DFTAdvisor
to share with a single scan cell. The default maximum number of control points that DFTAdvisor will allow to share a single scan cell upon invocation is 16.

Not all control points can share a single scan cell. If the -Reconvergence option is enabled for the Setup Test_point Insertion command and the forward trace of any two control points intersect, they must use separate scan cells. This may result in less than the maximum number of control points sharing any given scan cell.

- **-OShare integer**

An optional switch and integer pair that you can only use with the Simulation literal to specify the maximum number of observe points you want DFTAdvisor to share with a single scan cell. The default maximum number of observe points that DFTAdvisor will allow to share a single scan cell upon invocation is 16.

Not all observe points can share a single scan cell. If the -Reconvergence option is enabled for the Setup Test_point Insertion command and the backward trace of any two observe points intersect, they must use separate scan cells. This may result in less than the maximum number of observe points sharing any given scan cell.

- **-PATterns integer**

An optional switch and integer pair that you can only use with the Simulation or Multiphase literal to specify how many patterns you want DFTAdvisor to simulate to determine controllability and observability of the test points. The recommended number of patterns is $2^n - 1$. This number is used to synthesize a minimal phase decoder in LBISTArchitect. If the number is not $2^n - 1$, it is automatically changed to the closest $2^n - 1$ number (either higher or lower) and a warning is issued. The default number of patterns is 32767.

- **-PASs_size integer**

An optional switch and integer pair that you can only use with the Simulation literal to specify the number of test points added between each simulation analysis pass on the design. By default, DFTAdvisor adds 10 test points to the design between each simulation pass. This occasionally wastes a subset of these points, because controllability is improved by the addition of the first few test points in the set of ten and no new simulation is performed until after all ten are added.
Reducing the number of test points added per pass increases the number of simulation analysis passes needed and can reduce the number wasted test points. While increasing the number of test points added per pass reduces the number of passes, but can increase the number of wasted test points.

- **-Test_coverage percent**
  An optional switch and floating point number you can only use with the Multiphase literal to specify the percentage of target fault coverage. Once DFTAdvisor reaches your target coverage it ceases to add test points. The default value upon invocation of DFTAdvisor is 100.0.

- **-PHases integer**
  An optional switch and integer you can only use with the Multiphase literal to specify the number of phases into which you want DFTAdvisor to partition the entire test. The integer value must be in a power of 2. For example, 2, 4, or 8. Two phases is recommended for the start of analysis. Four and eight phase runs should be tried if the design is relatively large. In most cases, the later phases have decreased benefits and therefore, higher number of phases should be tried only if very high test coverage is required (greater than 99%). If you increase the number of phases, you must also increase the number of patterns (if greater than 32767 patterns). The default value upon invocation of DFTAdvisor is 4.

- **-BPC_threshold integer**
  An optional switch and integer you can only use with the Multiphase literal to specify the minimum benefit-per-cost (BPC) threshold. That is, the minimum number of faults that a test point should detect. DFTAdvisor uses this during the selection of control and observe points. For DFTAdvisor to select a test point, the test point must detect at least this number of faults. It can be useful to set this parameter low (1 or 2) in the start of the analysis run to see how the control points are being picked up. However, a low BPC threshold will cause the selection process to pick more control points in early phases and saturate the remaining faults. The default value upon invocation of DFTAdvisor is 5.

- **-Sig_prob_threshold percent**
  An optional switch and floating point number you can only use with the Multiphase literal to specify the minimum percentage of patterns that cause a circuit node to become 0 or 1. For a value of 15% enter “0.15”. DFTAdvisor
checks this during the preliminary analysis to determine control point candidates. Nodes for which a 0 or 1 pattern percentage falls below the threshold are called either zero-failing or one-failing, respectively. DFTAdvisor selects these zero-failing and one-failing nodes as candidates for control test point insertion. When the remaining fault list is still relatively large, increasing this percentage can have the effect of better control point selection. The default value upon invocation of DFTAdvisor is 10%. The maximum value is 25%. Using the default (10%) for 1000 patterns simulated, 100 of the 1000 patterns must be able to control a gate to 0 or 1, otherwise the node is considered a candidate.

• -NUm_detections integer

An optional switch and integer you can only use with the Multiphase literal to specify the confidence level that inserting a test point will detect a fault using random/BIST patterns. The integer range is 1 to 6. The default value upon invocation of DFTAdvisor is 4.

• -OP_cost multiplier

An optional switch and floating point number multiplier you can only use with the Multiphase literal to specify the cost of implementing an observe point rather than a new control point. This observe point cost is calculated as a multiple of a new control point. (Each control point is either a 2-input AND or OR gate.) Only gates with at least (BPCThreshold * OPCost) estimated faults are considered. DFTAdvisor uses this cost multiple to determine the overall value of inserting observe points. This integer should be specified according to your design environment and whether each observe point is implemented by a scan cell (high cost) or primary output (low cost). The default value upon invocation of DFTAdvisor is 4.0.

• -Rcp_cost multiplier

An optional switch and floating point number multiplier you can only use with the Multiphase literal to specify the cost of re-enabling a existing control point inserted in a previous phase verses adding a new control point. This cost of re-enabling is calculated as a multiplier of a new control point which has a cost of 1. An additional 2-input AND or OR gate is added each time an AND or OR control point is reactivated. The default value upon invocation of DFTAdvisor is 1.0.
Examples

The following example shows the flow of having DFTAdvisor automatically identify and insert two test points for controllability:

```bash
set system mode dft
setup scan identification none
setup test_point identification -control 2
run
// Performing test_point identification ...
// Number of control points to be identified = 2
// Number of observe points to be identified = 0
insert test logic -test_point on
report test points
Control [Selected]: /CNTR/U783/ZN or2a test_cntl1
Control [Selected]: /ADDR/U23/D1 or2a test_cntl2
```

The following example shows the flow for multiphase test point selection.

```bash
setup test_point identification -control 6 -observe 2 -patterns 1023
  -base multiphase -verbose -test_coverage 95.0 -phases 4
  -bpc_threshold 7 -sig_prob_threshold 0.05 -num_detections 1
// The # of patterns for the 4 phases are: 255 256 256 256
// The # of requested control points for the phases are: 0 2 2 2
set system mode dft
.
run
// Performing test_point identification ...
// Number of control points to be identified = 6
// Number of observe points to be identified = 2
.
report test points
Control [Selected]: /G_G364/out OR test_cntl_5
Control [Selected]: /G_G893/out AND test_cntl_4
Control [Selected]: /G_G1032/out AND test_cntl_3
Control [Selected]: /G_G1706/out AND test_cntl_2
Control [Selected]: /G_G295/out AND test_cntl_1
Control [Selected]: /G_G1513/out AND test_cntl_0
Observe [Selected]: /G_G1553/out test_obs
Observe [Selected]: /G_G1020/out test_obs
```
Setup Test_point Identification

Command Dictionary

Related Commands

Add Cell Models  Report Test Points
Add Test Points  Setup Scan Identification
Insert Test Logic  Setup Scan Insertion
Setup Test_point Insertion

Scope: All modes

Prerequisites: You must identify the modelname with the Add Cell Models command before using the -Model switch.

Usage

SETup TEst_point INsertion [-Control input_pin_name] [-Observe output_pin_name] [-None | -Model modelname] [-REconvergence {OFf | ON}]

Description

Specifies how DFTAdvisor configures the inputs for the system-defined control test points and the outputs for the system-defined observe test points.

The Setup Test_point Insertion command modifies how the Insert Test Logic command inserts test points. By default, the Insert Test Logic command creates a primary input for the control test points named “test_cntrl” and a primary output for the observe test points named “test_obs”.

If you want DFTAdvisor to automatically identify test points you can use the Setup Scan Identification command in combination with the Setup Test_point Identification and Run commands. After DFTAdvisor identifies the optimum test points, you then insert those test points with the Insert Test Logic command.

Arguments

- -Control input_pin_name

An optional switch and string pair that specifies the name of the control input pin at which you want to insert the test point. If you use the -None switch (the default) in combination with this switch, DFTAdvisor controls the test point it inserts with the pin specified by the “prefix” name, input_pin_name, as shown in Figure 2-6.

If you use the -Model switch in combination with this switch, DFTAdvisor controls the test point by adding an additional scan cell when the test logic is synthesized. The input_pin_name specifies the clock pin that controls the scan cell.
- **Observe** `output_pin_name`

   An optional switch and string pair that specifies the name of the observe output pin at which you want to insert the test point. If you use the `-None` switch (the default) in combination with this switch, DFTAdvisor controls the test point it inserts with the pin specified by the “prefix” name, `output_pin_name`, as shown in **Figure 2-7**.

   If you use the `-Model` switch in combination with this switch, DFTAdvisor controls the test point by adding an additional scan cell when the test logic is synthesized. The `output_pin_name` specifies the clock pin that controls the scan cell.

---

**Figure 2-6. Control Point Example**

**Figure 2-7. Observe Point Example**
- **None**

  An optional switch that specifies for DFTAdvisor to insert only the test point; without inserting an additional scan cell. This is the default.

- **-Model modelname**

  An optional switch that specifies for DFTAdvisor to insert a cell along with the test point. The specified model must be of type SCANCELL. You must identify the type of the `modelname` with the Add Cell Models command or have the type assigned in the library model before using this switch.

- **-REconvergence {Off | ON}**

  An optional switch and literal pair that enables the sharing of test points based on reconvergence analysis. Reconvergence analysis is performed by DFTAdvisor to determine which test points can share the same pin and scan cell. If the backward cones of observe points or the forward cones of control points intersect, the test points are not shared. When this option is On, the amount of sharing is limited, which guarantees that no fault masking can occur due to sharing (often a negligible phenomenon). The default is Off.

### Examples

The following example shows the flow of having DFTAdvisor automatically identify and insert two test points for controllability:

```plaintext
set system mode dft
setup scan identification none
setup test_point identification -control 2
run
// Performing test_point identification ...
// Number of control points to be identified = 2
// Number of observe points to be identified = 0
// 1: CV1=16458424 gate_index=3805 INV /CNTR/U783/ZN
// 2: CV1=16458417 gate_index=1058 BUF /ADDR/U23/D1

add cell models dffslp -type scancell CK D SDI SE
add cell models or2a -type Or
add cell models and2a -type and
setup test_point insertion -control test_cntrl1 -model dffslp
insert test logic -test_point on
report test points
  Control [Selected]: /CNTR/U783/ZN and2a test_cntrl1
  Control [Selected]: /ADDR/U23/D1 or2a test_cntrl1
```

DFTAdvisor Reference Manual, v8.6_4  
2-397
Related Commands

Add Cell Models  Setup Scan Identification
Insert Test Logic  Setup Test_point Identification
Setup Tied Signals

Scope: Setup mode

Usage

SETup TIed Signals 0 | 1 | X | Z

Description

Changes the default value for floating pins and floating nets which do not have assigned values.

The Setup Tied Signals command specifies the default value that the tool ties to all floating nets and floating pins that are not specified with the Add Tied Signals command. Upon invocation of the tool, if you do not assign a specific value, the tool assumes the default value is unknown.

If the model is already flattened and then you use this command, you must delete and recreate the flattened model.

Arguments

• 0
  A literal that specifies to tie the floating nets or pins to logic 0 (low to ground).

• 1
  A literal that specifies to tie the floating nets or pins to logic 1 (high to voltage source).

• X
  A literal that specifies to tie the floating nets or pins to unknown.

• Z
  A literal that specifies to tie the floating nets or pins to high-impedance
Examples

The following example ties floating net vcc to logic 1 and ties the remaining unspecified floating nets and pins to logic 0, then performs a scan identification run.

```plaintext
setup tied signals 0
add tied signals 1 vcc
set system mode dft
run
```

Related Commands

- Add Tied Signals
- Delete Tied Signals
- Report Tied Signals
**System**

Scope: All modes

**Usage**

SYStem `os_command`

**Description**

Passes the specified command to the operating system for execution.

The System command executes one operating system command without exiting the currently running application.

**Arguments**

- `os_command`
  
  A required string that specifies any legal operating system command.

**Examples**

The following example performs a scan identification run, then displays the current working directory without exiting DFTAdvisor:

```
set system mode dft
run
system pwd
```
Undo Display

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

Usage

UNDo DIisplay [level]

DFTInsight Menu Path:
- Display > Undo > One Level | N Levels

Description

Restores the previous schematic view.

The Undo Display command reverts the DFTInsight schematic view to the previous schematic that you specify. DFTInsight maintains a history of each time the schematic view changes up to the maximum history level. The Undo Display command allows you to restore these schematic views.

The maximum history level is 19. You can nullify Undo Display commands by using the Redo Display command.

Arguments

- level
  
  An integer that specifies the number of previous schematic views to which you want the DFTInsight schematic view to revert. The default is 1.

Examples

The following series of examples show how to display several different gate path schematics, each overwriting the last and then how to undo and redo the schematic displays. The first example invokes DFTInsight, then displays four custom gate paths by specifying the first and last gate identification numbers for each path (51 and 65):

  open schematic viewer
  add display path 23 51
The DFTInsight schematic view now displays all the gates between gate 65 and gate 102.

The next example undoes the last three schematic displays and restores the schematic view display of all the gates between gate 23 and gate 51:

```
undo display 3
```

The final example redoes (or nullifies) the last two undo operations and restores the schematic view display of all the gates between gate 51 and gate 65:

```
redo display 2
```

**Related Commands**

- Open Schematic Viewer
- Redo Display
Unmark

Tools Supported: DFTAdvisor and DFTInsight
Scope: All modes
Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

Usage

UNMark \{gate_id# | pin_pathname | instance_name\}... |-All | -Selected
DFTInsight Menu Path:
  Display > Unmark > All | Selected

Description

Removes the highlighting from the specified objects in the Schematic View window of objects.

The Unmark command unmarks objects in the DFTInsight Schematic View window by removing their graphical highlighting. You can unmark either all the objects in the design, individual objects that you specify, or all objects in the current selection list.

Arguments

- **gate_id#**
  A repeatable integer that specifies the gate identification number of the objects to unmark. The value of the gate_id# argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- **pin_pathname**
  A repeatable string that specifies the name of a pin whose gate you want to unmark.

- **instance_name**
  A repeatable string that specifies the name of the instance to unmark.
- **All**
  A switch that specifies to unmark all the gates in the design.

- **Select**
  A switch that specifies to unmark all the gates in the current selection list.

**Examples**

The following example specifies to unmark two objects:

```
unmark /i$142/q /i$141/q
```

**Related Commands**

- Mark
- Select Object
- Unselect Object
**Unselect Object**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

**Usage**

UNSelect OBject \{\texttt{gate\_id#} | \texttt{pin\_pathname} | \texttt{instance\_name}\}... \textbar \texttt{-All}

DFTInsight Menu Path:
- Display > Selection > Unselect All

**Description**

Removes the specified objects from the selection list in the DFTInsight schematic view.

The Unselect Object command unselects either all the objects in the schematic view or the individual objects that you specify.

**Arguments**

- \texttt{gate\_id#}
  
  A repeatable integer that specifies the gate identification number of the objects to unselect. The value of the \texttt{gate\_id#} argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- \texttt{pin\_pathname}
  
  A repeatable string that specifies the name of a pin whose gate you want to unselect.

- \texttt{instance\_name}
  
  A repeatable string that specifies the name of the instance to unselect.

- \texttt{-All}
  
  A switch that specifies to unselect all the gates in the design.
Examples

The following example specifies to unselect one object and then remove two more objects from the selection list:

```
unselect object /i$144/q
unselect object /i$142/q /i$141/q
```

Now all three objects are unselected.

Related Commands

- Mark
- Select Object
- Unmark
View

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

Usage

VIEW {gate_id# | pin_pathname | instance_name}... | -Selected | -Marked | -All

DFTInsight Menu Path:
Display > View > ...

Description

Displays the specified object in the DFTInsight Schematic View window.

The View command displays either the objects that you specify, the objects currently selected or marked, or all the objects in the display list.

Arguments

- **gate_id#**
  
  A repeatable integer that specifies the gate identification numbers of the objects to display. The value of the gate_id# argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- **pin_pathname**

  A repeatable string that specifies the name of a pin whose gate you want to display.

- **instance_name**

  A repeatable string that specifies the name of the instance to display.

- **-Selected**

  A switch that specifies to display all the gates that were selected using the Select Object command.
• **Marked**
  A switch that specifies to display all the gates that were marked using the Mark command.

• **-ALl**
  A switch that specifies to display all the gates in the display list.

**Related Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>Zoom In</td>
</tr>
<tr>
<td>Select Object</td>
<td>Zoom Out</td>
</tr>
<tr>
<td>View Area</td>
<td></td>
</tr>
</tbody>
</table>
**View Area**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

**Usage**

VIEW AREA \( x_1, y_1, x_2, y_2 \)

DFTInsight Menu Path:
Display > View > Area

**Description**

Displays the specified area in the DFTInsight Schematic View window.

The View Area command displays in the Schematic View window a rectangular area whose diagonal coordinates you specify. When you use this command, it adjusts both the horizontal and vertical axes.

The interactive method for viewing a specific area in the Schematic View window is to click on the View Area button at the top of the window and then using the click-drag-release mouse action, select the area that you want to view. This action transcripts the View Area command with the actual x,y coordinates for replayability purposes. You can cancel the View Area click-drag-release mouse action by pressing the Escape key prior to releasing the mouse button.

When entering the View Area command you must include the commas between each x and y coordinate.

**Arguments**

- \( x_1, \)

A required integer and comma specifying one x-coordinate of the rectangular area that you want to view. This x-coordinate is paired with the \( y_1 \) argument to define one corner of the rectangle.
• $y_1$,
  A required integer and comma specifying one y-coordinate of the rectangular area that you want to view. This y-coordinate is paired with the $x_1$ argument to define one corner of the rectangle.

• $x_2$,
  A required integer and comma specifying the x-coordinate of the opposite corner from $x_1$ of the rectangular area that you want to view. This x-coordinate is paired with the $y_2$ argument to define the corner of the rectangle diagonal from the $x_1$ and $y_1$ coordinates.

• $y_2$
  A required integer specifying the y-coordinate of the opposite corner from $y_1$ of the rectangular area that you want to view. This y-coordinate is paired with the $x_2$ argument to define the corner of the rectangle diagonal from the $x_1$ and $y_1$ coordinates.

Examples

To view a portion of the Schematic View window, click on the View Area button located at the top of the window.

Position the cursor pointer in one corner of the area that you want to view. Next, press the Select mouse button and drag the mouse to create a rectangle around the area you want to view. To display the area within the dynamic rectangle, release the Select mouse button.

Related Commands

View  
Zoom In  
Zoom Out
Write Atpg Setup

Scope: Dft mode

Prerequisites: You must insert the scan chains before using this command.

Usage

WRIted ATpg Setup basename [-Replace]

Description

Writes the test procedure and the dofile for inserted scan chains to the specified files.

The Write Atpg Setup command writes the test procedure file and the dofile file for the inserted scan chains. You can use these files to identify the scan chains when loading the scan design into FastScan or FlexTest.

Arguments

• basename

A required string that specifies the root name that you want DFTAdvisor to use when creating the test procedure and dofile. For the test procedure file, DFTAdvisor appends “.testproc” to the basename that you provide. For the dofile, DFTAdvisor appends “.dofile” to the basename.

• -Replace

An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists. The default upon invocation is not to replace the file.

Examples

The following example writes the test procedure, the dofile, and the new netlist for the inserted scan chains to the specified filenames:

```
add clocks 1 clk1
add clocks 0 clk0
set system mode dft
run
insert test logic
```
write atpg setup scan -replace
write netlist scan.edif -edif

Related Commands

Insert Test Logic            Write Netlist
Write Bist Setup

Scope: Dft mode

Prerequisites: You should enter all setup commands before executing this command.

Usage

Write Bist Setup [module_filename] [dofile_filename] [-Replace]
[-VHdl | -VERilog]

Description

Writes the top-level design interface and dofile used in LBISTArchitect.

The Write Bist Setup command writes the top-level interface of the design and the dofile used by LBISTArchitect. You need these files when you plan to run LBISTArchitect after scan insertion with DFTAdvisor.

The top-level design interface file describes the input/output pin instances of the design. The pin instances DFTAdvisor writes to this file are the same as the ones DFTAdvisor reads from the top-level VHDL entity or Verilog module.

The dofile contains LBISTArchitect commands that specify the internal scan chains, set up the LBISTArchitect controller, optionally set up multi-phase test points, run LBISTArchitect, and save LBISTArchitect results.

Note

If you execute this command after switching back to Setup mode some information is lost and not written to the dofile.

Arguments

- module_filename
  An optional string that specifies the name of the file to which DFTAdvisor writes the top-level design interface. The default filename upon invocation is bist_in.vhd (VHDL) or bist_in.v (Verilog) depending on the specified HDL type.
• `dofile_filename`

An optional string that specifies the name of the file to which DFTAdvisor writes the dofile. The default upon invocation is `bist.dofile`.

• `-Replace`

An optional switch that specifies for DFTAdvisor to replace the contents of the files, if the files already exist. The default upon invocation is not to replace the files.

• `-VHdl | -Verilog`

An optional switch specifying to write the design interface in the VHDL or Verilog format. The VHDL format is described under “Using VHDL” in the Design-for-Test Common Resources Manual. The default upon invocation is VHDL.

**Examples**

The following example performs all the setup, then writes the design interface and dofile files:

```
add scan groups grp1 s9234.testproc
add scan chains chain1 grp1 scan_in1 scan_out1
add clocks 0 clock
set capture clock clock
setup test_point identification -control 6 -observe 2 -patterns 1023
   -base multiphase -verbose -test_coverage 95.0 -phases 4
   -bpcthreshold 7 -sigprobthreshold 0.05 -num_detections 1
set system mode dft
setup scan identification none
run
add cell models or2a -type or
add cell models n1l -type inv
add cell models and2a -type and
add cell models fd1sqa -type sdff CP D
setup scan insertion -sen scan_en
insert test logic -test_point on -scan off
write netlist s9234_1_scan_mptp.gn -genie -replace
write bist setup bist_entity.vhd bist.dofile -replace -vhdl
```
Write Loops

Scope: Dft mode

Usage

WRIte LOops filename [-Replace]

Description

Writes a list of all loops to the specified file.

The Write Loops command writes all loops in a circuit to a file. For each loop, the report indicates whether the loop was broken by duplication. Loops that are not broken by duplication are shown as being broken by a constant value, which means the loop is either a coupling loop or has a single multiple fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

You can display the loops report information to the transcript by using the Report Loops command.

Arguments

• filename
  A required string that specifies the name of the file to which DFTAdvisor writes the loop report information.

• -Replace
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.

Examples

The following example writes a list of all loops to a file:

```
set system mode dft
write loops loop.info -replace
```

Related Commands

Report Loops
**Write Netlist**

Scope: All modes

**Usage**

WRItNeTlist *filename* [-Edif | -Tdl | -Verilog | -VHdl | -Genie | -Ndl | -Model] [-Replace]

**Description**

Writes the new netlist to the specified file.

The Write Netlist command writes the netlist which is either the one read into the system when you invoked DFTAdvisor, or the one created by the scan insertion process. If you do not specify one of the netlist format options, then by default DFTAdvisor uses the format that you specified when you invoked DFTAdvisor. For more information about the DFTAdvisor invocation options refer to “Shell Commands” on page 3-1.

It is recommended that you write the same netlist format as you read in, as DFTAdvisor is not intended to be a robust netlist translation tool.

**Arguments**

- *filename*
  A required string that specifies the name of the file to which DFTAdvisor writes the netlist.
- -Edif
  An optional switch specifying to write the netlist in the EDIF format.
- -Tdl
  An optional switch specifying to write the netlist in the TDL format.
- -Verilog
  An optional switch specifying to write the netlist in the Verilog format.
Write Netlist Command Dictionary

- **-VHdl**
  An optional switch specifying to write the netlist in the VHDL format as supported by DFTAdvisor and described under “Using VHDL” in the Design-for-Test Common Resources Manual.

- **-Genie**
  An optional switch specifying to write the netlist in the Genie format.

- **-Ndl**
  An optional switch specifying to write the netlist in the NDL format.

- **-Model**
  An optional switch specifying to generate an ATPG library given any Verilog file.

- **-Replace**
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.

**Examples**
The following example writes the modified netlist in Verilog format to a file:

```
add clocks 0 clock
set system mode dft
setup scan identification sequential atpg -percent 50
run
insert test logic -max_length 10
write netlist verilog.scan -verilog
```

**Related Commands**

Write Atpg Setup
Write Primary Inputs

Scope: All modes

Usage

WRItte PRimary Inputs *filename* [-Replace] [-All | *primary_input_pin*...]

Description

Writes primary inputs to the specified file.

The Write Primary Inputs command writes a list of either all the primary inputs of a circuit or a specific list of primary inputs that you specify into a file where it can be reviewed.

This command is identical to the Report Primary Inputs command except the data is written into a file.

Arguments

• *filename*
  A required string that specifies the name of the file to which DFTAdvisor writes the primary inputs.

• -Replace
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.

• -All
  An optional switch that specifies to write all the primary inputs to the file. This is the default.

• *primary_input_pin*
  An optional repeatable string that specifies a list of primary input pins that you want to write to the file.

Examples

The following example writes all primary inputs to a file:

```
write primary inputs inputfile
```
Related Commands

Report Primary Inputs
Write Primary Outputs

Scope: All modes

Usage

WRIte PRimary Outputs *filename* [-Replace] [-All | *primary_output_pin*...]

Description

Writes primary outputs to the specified file.

The Write Primary Outputs command writes a list of either all the primary outputs of a circuit or a specific list of primary outputs that you specify into a file where it can be reviewed.

This command is identical to the Report Primary Outputs command except the data is written into a file.

Arguments

- *filename*
  A required string that specifies the name of the file to which DFTAdvisor writes the primary outputs.
- -Replace
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.
- -All
  An optional switch that specifies to write all the primary outputs to the file. This is the default.
- *primary_input_pin*
  An optional repeatable string that specifies a list of primary output pins that you want to write to the file.

Examples

The following example writes all primary outputs to a file:

```
write primary outputs outputfile
```
Write Primary Outputs

Related Commands

Report Primary Outputs
Write Scan Identification

Scope: All modes

Usage

WRIt e SCan Identification *filename* [-Replace] [-Full | -Identified | -Defined]
 [-DOfile | -Backannotation]

Description

Writes a list of the scan instances which DFTAdvisor has identified or you have defined as scan cells.

The Write Scan Identification command writes scan cell instances that either DFTAdvisor identified during the identification process or that you defined by using the Add Scan Instances or Add Scan Models commands. The Write Scan Identification command lists the scan instances in descending order, with the first instance being the most critical scan instance.

If you are identifying scan sequential, the Write Scan Identification command displays the sequential loops that DFTAdvisor cut after you have performed the identification process with the Run command.

If you are identifying partition scan, the Write Scan Identification command displays the partition cells that DFTAdvisor flagged during the identification process that you perform with the Run command.

If you issue the command without specifying -Backannotation or -Dofile options, the command writes both identified and defined scan instances in a format that is similar to the output of the Report Scan Identification command.

Arguments

• *filename*
  A required string that specifies the name of the file to which DFTAdvisor writes the scan instances.

• -Replace
  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.
An optional switch that specifies at what level to write the file. The -Full option specifies to write both identified and defined scan instances. This is the default. The -Identified option specifies to write only scan instances that DFTAdvisor identified during the identification process. The -Defined option specifies to write only scan instances that you defined by using the Add Scan Instances or Add Scan Models commands.

An optional switch that specifies to write the scan instances in dofile or back annotation format. In dofile format the file is written as a list of lines which can be executed by the Dofile command. For example:

```
add scan instances instance_pathname
```

### Examples

The following example writes all scan instances to a file after performing a full scan identification run:

```
set system mode dft
setup scan identification full_scan
run
write scan identification scanfile -identified
```

The following is an example of the files contents:

<table>
<thead>
<tr>
<th>Type</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>defined</td>
<td>/I_1_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_1_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_6_16</td>
</tr>
<tr>
<td>identified</td>
<td>/I_8_16</td>
</tr>
</tbody>
</table>

### Related Commands

Report Scan Identification
Write Subchain Setup

Scope: Dft mode

Usage

WRIte SUbchain Setup filename -Replace

Description

Writes the appropriate Add Sub Chains commands to a file so that DFTAdvisor can understand the preexisting scan sub-chains at the top-level of this module.

The Write Subchain Setup command is useful if you are performing a block-by-block test synthesis at the lower design level. By using this command to specify for DFTAdvisor to write the sub-chain setup information at the lower-level and then reading in the setup file as part of the scan sub-chain setup, you can avoid having to define the preexisting scan sub-chain at the higher design level with the Add Sub Chains command.

Arguments

- **filename**
  
  A required string that specifies the name of the file to which DFTAdvisor writes the scan sub-chain setup information. For information on this file format, refer to the Insert Test Logic command.

- **-Replace**

  An optional switch that specifies for DFTAdvisor to replace the contents of the file, if the file already exists.

Examples

The following example writes the scan chain setup information for this module so that you can later use the *module1.setup* file at the top-level to define the preexisting scan sub-chain:

```
insert test logic
write subchain setup /user/designs/module1.setup
```
The following example shows the contents of the `module1.setup` file:

```plaintext
add sub chains /user/designs/sub1 subc1 scan_in1 scan_out1 7 mux_scan scan_en
```

**Related Commands**

- Add Sub Chains
- Report Sub Chains
- Insert Test Logic
**Zoom In**

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

**Usage**

```
ZOOm IN scale_factor
```

DFTInsight Menu Path:
   Zoom > In (Common popup menu)

**Description**

Enlarges the objects in the DFTInsight Schematic View window by reducing the displayed area.

**Arguments**

- `scale_factor`

  A required integer or real number greater than 1.0 specifying the multiplication factor that DFTInsight uses to determine how much to enlarge the objects.

**Examples**

The following example zooms in the Schematic View window so that objects are twice as big:

```
zoom in 2
```

**Related Commands**

- Zoom Out
Zoom Out

Tools Supported: DFTAdvisor and DFTInsight

Scope: All modes

Prerequisites: You must first invoke the optional DFTInsight application and have it displaying instances.

Usage

ZOOM OUT \( scale \_factor \)

DFTInsight Menu Path:
   - Zoom > Out (Common popup menu)

Description

Reduces the objects in the DFTInsight Schematic View window by increasing the displayed area.

Arguments

- \( scale \_factor \)

   A required integer or real number greater than 1.0 specifying the division factor that DFTInsight uses to determine how much to reduce the objects.

Examples

The following example zooms out the Schematic View window so that objects are one-third as large:

   zoom out 3

Related Commands

   Zoom In
Chapter 3
Shell Commands

Shell Command Descriptions

The following subsection documents the shell command that you use to invoke the command-line version of DFTAdvisor.

The notational conventions used here are the same as those used in other parts of the manual. Do not enter any of the special notational characters (such as, {}, [], or |) when typing the command. For a complete description of the notational conventions used in this manual, refer to Command Line Syntax Conventions on page xvi in the About This Manual.
**dftadvisor**

Prerequisites: You must have a netlist in one of the required formats and a cell library containing descriptions of all the cells used in the design. The valid netlist formats are: EDIF, TDL, VHDL, Verilog, Genie, and Spice.

**Usage**

\[
dftadvisor \\
OR \\
dftadvisor \{design\_name \{ -Edif | -Tdl | -VHdl | -VERIlog | -Genie | -SPice \} \{ -LIBrary filename \} \{-SEnsitive\} \{-LOG filename \} \{-Replace\} \{-TOp module\_name \} \{-Dofile dofile\_name \} \{-LICense retry\_limit \} \{-NOGui\} \}
\]

**Description**

You can invoke DFTAdvisor in one of two ways. Using the first option, you enter just the application name on the shell command line. Once the tool is invoked, a dialog box prompts you for the required arguments (design_name, design type, and library). Browser buttons are provided for navigating to the design and library. Once the design and library are loaded, the tool is in Setup mode, ready for you to begin working on your design. Invoking DFTAdvisor in this way automatically starts a graphical session.

Using the second option requires you to enter all required arguments at the shell command line. When the tool is finished invoking a graphical session, the design and library are also loaded. The tool is now in Setup mode, ready for you to begin working on your design.

If you want to invoke DFTAdvisor in command-line mode, you must use the second invocation option with the -Nogui switch specified.

**Arguments**

- **design\_name**
  
  A required string that specifies the pathname of the design on which to invoke. The design must be in the format that you specify by using one of the following switches: -Edif, -Tdl, -Vhdl, -Verilog, -Genie, or -Spice.
• **-Edif**
  
  A switch specifying that the *design_name* is a netlist in EDIF format.

• **-TDI**
  
  A switch specifying that the *design_name* is a netlist in TDL format.

• **-VHdl**
  
  A switch specifying that the *design_name* is a netlist in VHDL format as supported by DFTAdvisor. You must also have a *dft.map* file present in the same directory as the VHDL netlist. For information on the format of the *dft.map* file and the supported VHDL constraints, refer to “Reading VHDL” in the *Design-for-Test Common Resources Manual*.

• **-VERIlog**
  
  A switch specifying that the *design_name* is a netlist in Verilog format.

• **-Genie**
  
  A switch specifying that the *design_name* is a netlist in Genie format. When reading in a directory-based Genie netlist (netlist hierarchy corresponds to UNIX directories), DFTAdvisor first tries to access the directory based on an uppercase directory name. If it is not successful in finding the directory, it then tries the original case of the instance names in the design.

• **-SPice**
  
  A switch specifying that the *design_name* is a netlist in Spice format.

• **-LIBrary filename**
  
  A required switch and string pair that specifies the name of the file containing the library descriptions for all cell models in the *design_name*. This argument is not required if the specified netlist fully defines all the primitives in the design (which can occur in Verilog and TDL formats). If the library does not contain scan models for any of the sequential element your design uses, DFTAdvisor issues a warning message stating this when you switch from Setup mode to Dft mode during the session.
-SEnsitive
A switch that specifies for DFTAdvisor to consider pin, instance, and net pathnames case sensitive. The default is case-insensitive. Regardless of the use of this switch, command names are always case insensitive.

-LOg filename
A switch and string pair that specifies the name of the file to which you want DFTAdvisor to write all session information. The default is to display session information to the standard output.

-Replace
A switch that specifies to overwrite the -Log filename if one by the same name already exists.

-TOp model_name
A switch and string pair that specifies the name of the top-level model in the netlist. If the netlist describes multiple top modules and you do not use this argument, DFTAdvisor assumes the first top to be the top module.

-Dofile dofile_name
An optional switch and string pair that specifies the name of the dofile that you want DFTAdvisor to execute upon invocation.

-LICense retry_limit
A switch and string pair that specifies the number of time to try to invoke DFTAdvisor before quitting. When this option is used in batch mode (-Nogui and -Dofile are specified), if all licenses are taken, you can specify this option and DFTAdvisor tries to get a license once every minute until either a license is available or the number of tries is exhausted. When this option is used in interactive mode, you are prompted to choose whether you want to try again or exit. If no license is found within the specified retry_limit, the invocation process aborts.

-NOGui
An optional switch that invokes the tool in command-line mode. The default is to invoke the graphical user interface (GUI).
• -Help
  An optional switch that only displays the version and usage line for DFTAdvisor. No other arguments can be specified with this switch.

• -VERSion
  An optional switch that only displays the version of the DFTAdvisor software that you currently have available. No other arguments can be specified with this switch.

Example

The following example invokes DFTAdvisor in command-line mode on an EDIF netlist named design1.edif. This design contains library parts that are specified in a file called mitsu_lib10. A session log is kept in a file called design1_scan.log, whose content is replaced if it already exists.

    $MGC_HOME/bin/dftadvisor design1.edif -edif -library
    mitsu_lib10 -log design1_scan.log -replace -nogui

The following example also invokes DFTAdvisor, but then has you use the invocation dialog box to enter the same arguments. This method invokes DFTAdvisor in graphical mode.

    $MGC_HOME/bin/dftadvisor
    Design: design1.edif
    Format: EDIF
    ATPG Library: mitsu_lib10
    Log File: design1_scan.log
    Overwrite Existing File: ON
INDEX

A
About This Manual, xiii
Acronyms, xvii

B
B command, 2-240

C
Command Dictionary, 2-1
Commands
  Add Atpg Constraints, 2-16
  Add Buffer Insertion, 2-18
  Add Cell Library, 2-21
  Add Cell Models, 2-22
  Add Clock Groups, 2-26
  Add Clocks, 2-28
  Add Display Instances, 2-30
  Add Display Loop, 2-34
  Add Display Path, 2-36
  Add Display Scanpath, 2-38
  Add Faults, 2-40
  Add LFSR Connections, 2-42
  Add LFSR Taps, 2-44
  Add LFSRs, 2-46
  Add Mapping Definition, 2-48
  Add Mos Direction, 2-51
  Add Mtpi Controller, 2-53, 2-143
  Add Mtpi Output, 2-55
  Add Net Property, 2-57
  Add Nofaults, 2-58
  Add Nonscan Instances, 2-60
  Add Nonscan Models, 2-62
  Add Notest Points, 2-64
  Add Output Masks, 2-66
  Add Pin Constraints, 2-68
  Add Pin Equivalences, 2-70
  Add Primary Inputs, 2-72
  Add Primary Outputs, 2-74
  Add Read Controls, 2-75
  Add Scan Chains, 2-76
  Add Scan Group, 2-78
  Add Scan Instances, 2-80
  Add Scan Models, 2-83
  Add Scan Pins, 2-85
  Add Seq_transparent Constraints, 2-88
  Add Sub Chains, 2-90
  Add Test Points, 2-94
  Add Tied Signals, 2-100
  Add Write Controls, 2-102
  Analyze Control Signals, 2-103
  Analyze Drc Violation, 2-105
  Analyze Input Control, 2-111
  Analyze Output Observe, 2-113
  Analyze Testability, 2-115
B, 2-240
  Close Schematic Viewer, 2-118
  Delete Atpg Constraints, 2-119
  Delete Buffer Insertion, 2-121
  Delete Cell Models, 2-123
  Delete Clock Groups, 2-125
  Delete Clocks, 2-127
  Delete Display Instances, 2-128
  Delete Faults, 2-130
  Delete LFSR Connections, 2-132
  Delete LFSR Taps, 2-134
  Delete LFSRs, 2-136
  Delete Mapping Definition, 2-138
  Delete Mos Direction, 2-142
  Delete Mtpi Output, 2-144
  Delete Net Property, 2-146
  Delete Nofaults, 2-147
  Delete Nonscan Instances, 2-149
  Delete Nonscan Models, 2-151
  Delete Notest Points, 2-153
  Delete Output Masks, 2-155
  Delete Pin Constraints, 2-157
  Delete Pin Equivalences, 2-159
  Delete Primary Inputs, 2-160
  Delete Primary Outputs, 2-162
  Delete Read Controls, 2-164
INDEX [continued]

Delete Scan Chains, 2-165
Delete Scan Groups, 2-166
Delete Scan Instances, 2-167
Delete Scan Models, 2-169
Delete Scan Pins, 2-171
Delete Seq_transparent Constraints, 2-172
Delete Sub Chains, 2-174
Delete Test Points, 2-176
Delete Tied Signals, 2-178
Delete Write Controls, 2-180
dftadvisor, 3-2
Dofile, 2-181
Exit, 2-183
Extract Subckts, 2-185
F, 2-241
Flatten Subckt, 2-186
Help, 2-187
Insert Scan Chains, 2-188
Insert Test Logic, 2-193
Mark, 2-200
Open Schematic Viewer, 2-202
Read Subckts Library, 2-203
Redo Display, 2-204
Report Atpg Constraints, 2-206
Report Buffer Insertion, 2-207
Report Cell Models, 2-208
Report Clock Groups, 2-210
Report Clocks, 2-211
Report Control Signals, 2-212
Report Dft Check, 2-214
Report Display Instances, 2-218
Report Drc Rules, 2-221
Report Environment, 2-227
Report Faults, 2-229
Report Feedback Paths, 2-233
Report Flatten Rules, 2-235
Report Gates, 2-239
Report LFSR Connections, 2-245
Report LFSRs, 2-246
Report Loops, 2-247
Report Mapping Definition, 2-248
Report Mos Direction, 2-251
Report Mtpi Controller, 2-252
Report Net Properties, 2-254
Report Nofaults, 2-255
Report Nonscan Instances, 2-257
Report Nonscan Models, 2-259
Report Notest Points, 2-261
Report Output Masks, 2-262
Report Pin Constraints, 2-263
Report Pin Equivalences, 2-265
Report Primary Inputs, 2-266
Report Primary Outputs, 2-267
Report Read Controls, 2-268
Report Scan Cells, 2-269
Report Scan Chains, 2-271
Report Scan Groups, 2-272
Report Scan Identification, 2-273
Report Scan Instances, 2-275
Report Scan Models, 2-276
Report Scan Pins, 2-277
Report Seq_transparent Constraints, 2-278
Report Statistics, 2-279
Report Sub Chains, 2-281
Report Test Logic, 2-282
Report Test Points, 2-284
Report Testability Analysis, 2-286
Report Tied Signals, 2-289
Report Write Controls, 2-290
Reset State, 2-291
Ripup Scan Chains, 2-292
Run, 2-294
Save Patterns, 2-296
Save Schematic, 2-299
Select Object, 2-300
Set Bist Initialization, 2-302
Set Capture Clock, 2-303
Set Contention Check, 2-305
Set Control Threshold, 2-307
Set Dofile Abort, 2-308
INDEX [continued]

Set Drc Handling, 2-309
Set Fault Sampling, 2-317
Set Flatten Handling, 2-318
Set Gate Level, 2-322
Set Gate Report, 2-324
Set Identification Model, 2-327
Set Instancename Visibility, 2-330
Set Internal Fault, 2-332
Set I0 Insertion, 2-333
Set Latch Handling, 2-336
Set Lockup Latch, 2-338
Set Logfile Handling, 2-341
Set Loop Duplication, 2-343
Set Multiple Scan_enables, 2-345
Set Net Resolution, 2-347
Set Nonscan Handling, 2-348
Set Observe Threshold, 2-349
Set Pattern Save, 2-350
Set Random Patterns, 2-351
Set Scan Type, 2-352
Set Schematic Display, 2-353
Set Screen Display, 2-356
Set Sensitization Checking, 2-357
Set Stability Check, 2-358
Set System Mode, 2-359
Set Test Logic, 2-361
Set Trace Report, 2-364
Set Zoom Factor, 2-365
Setup LFSRs, 2-366
Setup Output Masks, 2-368
Setup Pin Constraints, 2-370
Setup Scan Identification, 2-372
Setup Scan Insertion, 2-380
Setup Scan Pins, 2-385
Setup Test_point Identification, 2-387
Setup Test_point Insertion, 2-395
Setup Tied Signals, 2-399
System, 2-401
Undo Display, 2-402
Unmark, 2-404
Unselect Object, 2-406
View, 2-408
View Area, 2-410
Write Atpg Setup, 2-412
Write Bist Setup, 2-414
Write Loops, 2-416
Write Netlist, 2-417
Write Primary Inputs, 2-419
Write Primary Outputs, 2-421
Write Scan Identification, 2-423
Write Subchain Setup, 2-425
Zoom In, 2-427
Zoom Out, 2-428

D

dftadvisor command, 3-2
DFTInsight Commands
Add Display Instances, 2-30
Add Display Loop, 2-34
Add Display Path, 2-36
Add Display Scanpath, 2-38
Analyze Drc Violation, 2-105
Close Schematic Viewer, 2-118
Delete Display Instances, 2-128
Mark, 2-200
Open Schematic Viewer, 2-202
Redo Display, 2-204
Report Display Instances, 2-218
Report Feedback Paths, 2-233
Save Schematic, 2-299
Select Object, 2-300
Set Gate Level, 2-322
Set Gate Report, 2-324
Set Instancename Visibility, 2-330
Set Schematic Display, 2-353
Set Zoom Factor, 2-365
Undo Display, 2-402
Unmark, 2-404
Unselect Object, 2-406
View, 2-408
INDEX [continued]

View Area, 2-410
Zoom In, 2-427
Zoom Out, 2-428

F
F command, 2-241

I
Inputs of DFTA, 1-2

O
Outputs of DFTA, 1-2

R
Related documentation, xiv
Reporting
  report gate format, 2-239

S
Spice commands
  Add Mos Direction, 2-51
  Add Net Property, 2-57
  Delete Mos Direction, 2-142
  Delete Net Property, 2-146
  Extract Subckts, 2-185
  Flatten Subckt, 2-186
  Read Subckts Library, 2-203
  Report Mos Direction, 2-251
  Report Net Properties, 2-254

T
TIE0, scannable, 2-60
TIE1, scannable, 2-60